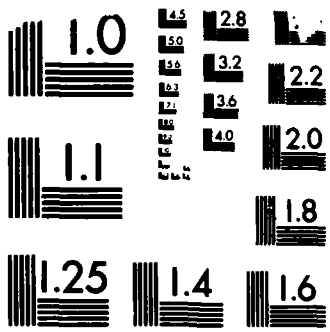
AD-A138 333 UNIVERSAL DISK CONTROLLER FOR MICROCOMPUTERS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AF OH SCHOOL OF ENGINEERING F N ELAM DEC 83 AFIT/GE/EE/83D-20 1/2

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNIVERSAL DISK CONTROLLER FOR MICROCOMPUTERS

THESIS

AFIT/GE/EE/83D-20

Frank N. Elam
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/83D-20

UNIVERSAL DISK CONTROLLER
FOR MICROCOMPUTERS

THESIS

AFIT/GE/EE/83D-20 Frank N. Elam
Capt USAF

AFIT/GE/EE/83D-20

UNIVERSAL DISK CONTROLLER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

A-1

by

Frank N. Elam, B.S.

Captain USAF

Graduate Electrical Engineering

December 1983

Preface

This project is dedicated to all microcomputer users that have little or no means of communicating with other microcomputers.

I would like to thank Major Lillie and Captain Baker for the advice they contributed. Gratitude is also due to all AFIT students who were eager to thoroughly test this project. This testing played an important role in producing an error free file transfer system.

TABLE OF CONTENTS

Page

Preface ii

List of Figures iv

List of Tables vi

Abstract vii

I. Introduction

Background I-1
Scope I-1
General Approach I-3

II. Analysis

Floppy Disks II-1
Floppy Disk Drives II-4
CP/M File System II-9

III. Hardware Design

Hardware Overview III-1
Miscellaneous Circuits III-2
Data Recovery Circuit III-5
Write Circuit III-9

IV. Software Design / Implementation

Software Overview IV-1
Library Routines IV-1
Systems Routines IV-16
File Manipulation Program IV-20

V. Recommendations and Conclusions

Bibliography BIB-1

Appendix A: 'C' Code Listing A-1

Appendix B: Assembly Code Listings B-1

List of Figures

Figure		Page
II-1	FM Encoding	II-3
II-2	MFM Encoding	II-3
II-3	Write Data to Read Data Comparisons	II-5
II-4	SA450 Functional Diagram	II-7
II-5	SA800/801 Functional Diagram	II-8
II-6	SA400 Interface Connections	II-10
II-7	SA800 Interface Connections	II-13
II-8	CP/M Directory	II-17
III-1	UDC Interface	III-2
III-2	10 Sectored Hard Sector Disk	III-5
III-3	IX/Sector Pulse Separator	III-6
III-4	IX/Sector Timing	III-7
III-5	Size/Density Decoder	III-9
III-6	Tx Clock	III-10
III-7	Multiplexer	III-12
III-8	Read Circuit Block Diagram	III-13
III-9	Read Timing	III-14
III-10	RCLK Separatior	III-16
III-11	RxClk Circuit	III-18
III-12	Write Circuit Block Diagram	III-20
III-13	Write Timing Diagram	III-22
III-14	Tx Data Pulser	III-22
III-15	Precompensation Circuit	III-24
III-16	Early/Late Timing	III-25

List of Figures

III-17 Early/Late State Diagram III-27

III-18 Boolean Reduction III-28

III-19 Early/Late Generator III-29

III-20 Data Shifter III-29

List of Tables

III-1 MUX Function Table III-12

III-2 Implementation Table III-12

III-3 State Table III-28

Abstract

A highly versatile floppy disk controller was designed and constructed. This disk controller was designed to allow maximum software interfacing to all Shugart 5" or 8" floppy disk drives.

A library of routines was wittten using the Z80 assembly language. These routines perform all functions necessary to control drive selection, read/write head movement, and data transfer. The data transfer routines are for use with frequency modulation data encoding and modified frequency modulation data encoding.

Three additional Z80 assembly language programs were designed for specific use with three micro computer systems: "IBM", "North Star Horizon", and "NEC 8000". These system routines are capable of reading and writing an unlimited number of contiguous disk sectors.

A file transfer program was written in the "C" programming language. This program in conjunction with the system routines and library routines is capable of transferring an unlimited number of CP/M files from one system to another.

I. Introduction

Background

The information contained within this thesis addresses a significant problem that almost every microcomputer user is faced with today. The problem is the inability to transfer files from one microcomputer system to another. A total and absolute solution to this problem is very close to an impossibility. However, an in depth investigation into this problem has turned up some interesting possibilities and partial solutions that make the problem as a whole, easier for microcomputer users to cope with.

The reason this problem has evolved is a simple one. Changes have occurred and are occurring so rapidly within the microcomputer industry that it is very difficult for the manufacturers to adhere to standards. If a standard is adopted, it does not take long for technology to advance to a point that makes the premise for the standard obsolete.

Scope

The research accomplished for this project was not aimed at solving the entire problem, but was focussed on an area which was manageable, and most likely to turn up useful results. The scope of the problem is outlined below.

1) Operating System - All files transferred must be

CP/M files. CP/M is an operating system for microcomputers produced by Digital Research. There are versions of CP/M for a wide variety of microcomputers. It can be used with almost any computer with an 8080 or Z80 microprocessor and has 5 1/4" and/or 8" disk drives. The introduction of the Microsoft SoftCard and add-on processing boards have extended the use of CP/M to Apple and Commodore computers. CP/M is the closest to a universal operating system there is for microcomputers. For this reason, CP/M was chosen to be the operating system used for this project.

2) Microcomputer - The microcomputer chosen to be the home for the disk controller is Advanced Digital's Super Quad Single Board Computer (SQ SBC) housed in a TT-10 S-100 bus system. The hardware developed in this project is designed to operate only on this computer. However, it is capable of transferring files between disks from a wide variety of 8080/Z80 based microcomputers.

3) Software - The software developed for the Universal Disk Controller (UDC) demonstrates the capabilities of the controller by transferring files between three microcomputer systems. The systems chosen were the home system (Super Quad), Northstar Horizon, and the NEC PC-8000. These systems were chosen due to their availability and varied formats.

4) Format Limitations - The 4 MHz CPU of the SQ SBC is not fast enough to allow 8 inch double density formats to be accomplished with the UDC. This fact will be discussed in more detail in a later chapter.

General Approach

The Universal Disk Controller was designed to be capable of transferring every clock bit and every data bit on a floppy disk to core memory. Absolutely no data formatting or decoding is accomplished by hardware. This technique provides maximum flexibility in transferring data between a floppy disk and the microcomputer. Any new disk formatting scheme a microcomputer manufacturer chooses to use can be accommodated by software updates only.

Along with this approach to data transfer comes the disadvantage of a lack of speed, and a higher core memory requirement. This disadvantage is of little significance because the disk controller and associated software is used only when files are being transferred for other systems. The SQ SBC has its own disk controller for its domestic file activities.

Order of Presentation

The following chapters discuss the research, design, and implementation of the Universal Disk Controller in the same order it progressed from beginning to end. Chapter 2

presents information acquired in pre-development research. It is an analysis of floppy disks, floppy disk drives, and the CP/M file system. Chapter 3 contains all information pertaining to the actual developement of hardware. It is divided into three sections: miscellaneous circuits, read circuits, and write circuits. Chapter 4 gives an in depth discussion of the implementation (software design) of the UDC. It will be presented in a bottom up fasion by discussing first how to operate the UDC and ending with the file transfer software. The last chapter will give recommendations for ways to improve the UDC, recommendations for follow on projects, and concluding remarks.

II. Analysis

Floppy Disks

1) Media - A floppy disk is a cylindrical magnetic media used to store information. These disks are normaly 8 inches or 5 1/4 inches in diameter. The 8 inch disk is called a "regular floppy" and the 5 1/4 inch disk is commonly referred to as a "mini floppy". [Zak - 79]

The floppy disk and its associated hardware have significant advantages over other means of data storage. These are low cost, high speed, and high capacity. These advantages make floppy disk systems very popular in the world of microcomputers. [Gib - 80]

Each diskette is housed in a flexible jacket. This jacket has the primary function of reducing faults caused by static electricity and physical contamination on the media. The inner lining of the jacket is made of a low friction lining to minimize wear. [Zak - 79]

2) Format - The data recorded on floppy disks is logically arranged in concentric bands called "tracks". Each track is further subdivided into blocks of data called "sectors". The technique of dividing disks into sectors allows random access to the recorded data (a block of data can be recovered without reading the entire track).

A disk controller must be able to distinguish the beginning of each sector. There are two methods used to do

this. The first (hard sectoring), uses a method of marking the beginning of each sector with a hole punched in the disk. The second method (soft sectoring) recognizes the beginning of a sector by a unique pattern of data recorded on the magnetic surface. In either case, most disks have at least one hole which may be used as an index to mark the beginning of the first sector. The arrangement of these blocks of information on a disk is called a "format". [Hoe - 80] [Nic - 81]

3) Information Encoding - There are several schemes used to record bits of information on the disk. The most popular schemes are frequence modulation (FM) and modified frequency modulation (MFM). MFM can pack bits into an area half the size required by the FM scheme. FM is a "single density" encoding scheme, while MFM is "double density". [Har - 79]

Figure II.1 depicts the FM encoding scheme. Every data cell starts with a clock pulse. The purpose of the clock pulse is to provide a timing reference and define exactly where a data pulse should occur. If a pulse occurs between clock pulses, this is recognized as a data "1". The absence of a pulse is a data "0". Each pulse can be written a minimum of 2 microseconds apart regardless of the size disk or encoding scheme. This means that the FM encoding scheme has one data bit for every four microseconds.

MFM encoding is used to reduce the number of clock

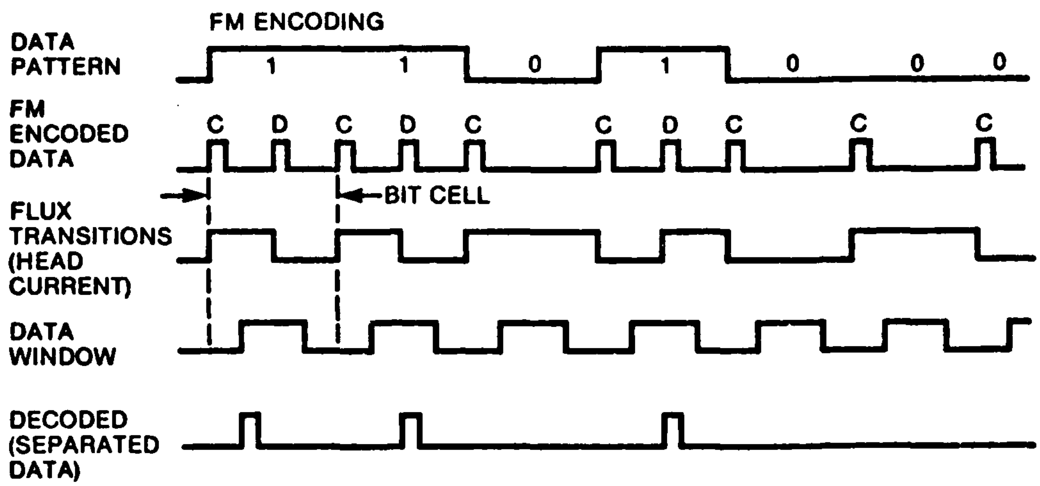


Figure II.1 FM Encoding [Shu A - 81]

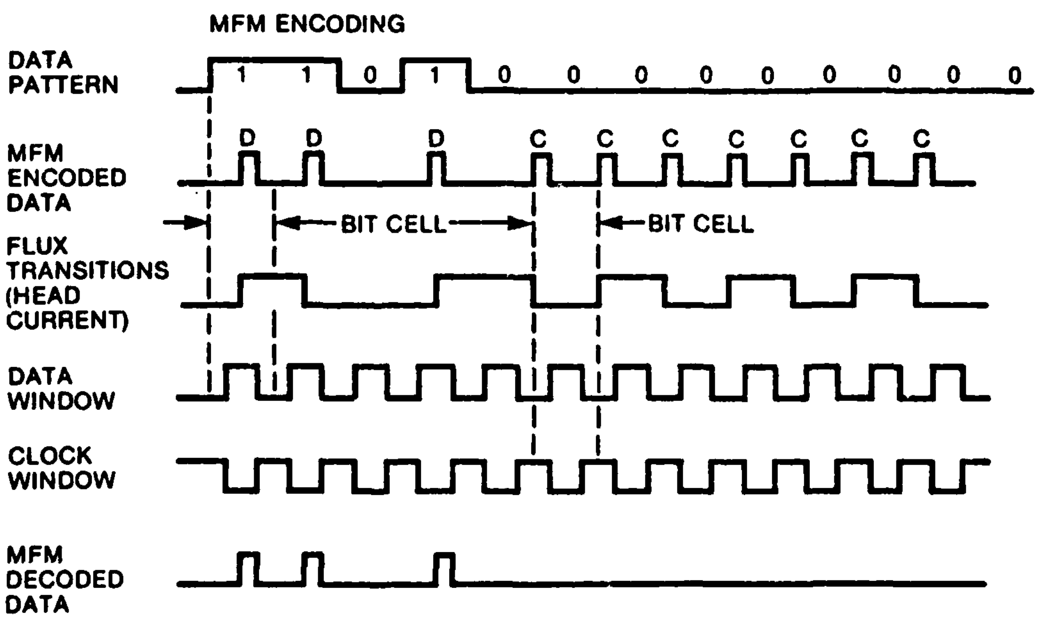


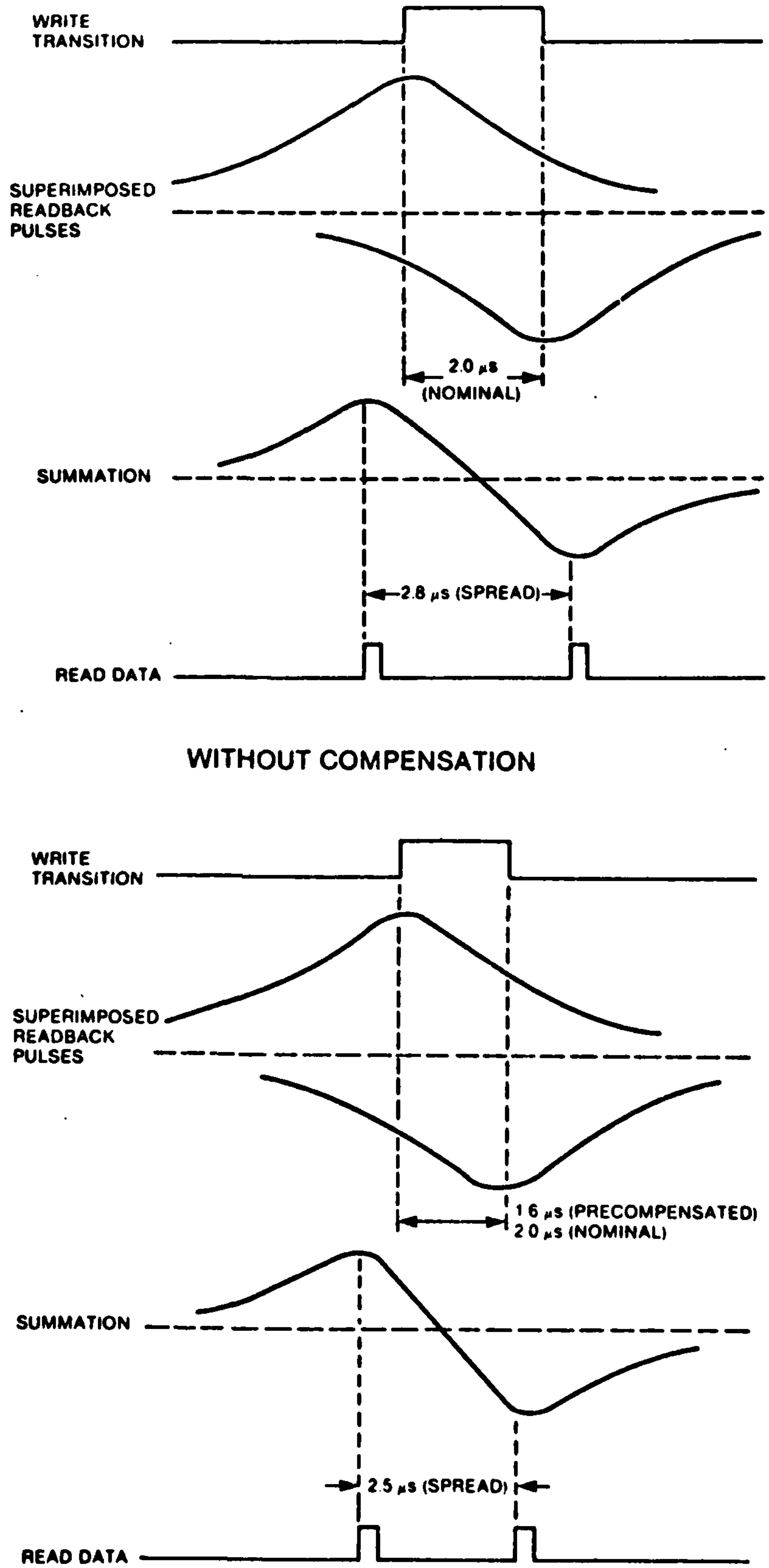
Figure II.2 MFM Encoding [Shu A - 81]

pulses which allows data bits to be packed in a higher density. Clock pulses are not written unless two consecutive "0" data bits occur, therefore each bit cell contains only one pulse (see figure II.2). If the criteria of 2 microseconds minimum between pulses is used, then the MFM encoding scheme can write two data bits for every four microseconds. This is twice the density of FM and hence the name "double density".

4) Write Precompensation - Recovering data written in a double density format is more critical than for single density. The primary reason for this is a phenomenon called "bit shift". When two pulses are written close together without surrounding pulses, they tend to spread apart (see figure II.3). This undesirable circumstance causes random frequence components to occur among the fundamental frequency of the recorded data. [Shu A - 81]

The primary component of bit shift is predictable which means a compensation factor can be applied to reduce its adverse affect. This compensation is called "write precompensation". Write precompensation is accomplished by writting pulses about .4 microseconds early or late (opposite the direction they tend to move). [Shu (A) - 81]

Write precompensation is most important on the inner tracks where bit density is highest. Generally speaking, write precompensation is desired starting at about the middle track and only with double density encoding. [Shu G - 81]



WRITE DATA TO READ DATA COMPARISONS WITH COMPENSATION

Figure II.3 [Shu B - 81]

Floppy Disk Drives

1) Drives - The disk drives used with the Universal Disk Controller are Shugart 5 1/4" (SA400) and 8" (SA801) drives. These were chosen due to high quality and their standard interface connections. A description of both models is given below.

The Shugart SA400 is a 5 1/4 inch floppy disk drive (see figure II.4). This is a single sided drive with the capability of reading and writing both single and double density. It uses a DC velocity servo spindle motor which rotates the disk at 300 rpm. The single head is positioned by a spiral cam mechanism. This mechanism gives the drive the capability of accessing 40 tracks. [Shu G - 81]

The Shugart SA801 is an 8 inch floppy disk drive (see figure II.5). This is a single sided drive with the capability of reading and writing both single and double density. The drive motor is an AC syncronous spindle motor which is synchronous with AC line frequency (360 rpm). Head movement is controlled in the same fasion as the smaller 5 1/4" drive. It can access 77 tracks which is the IBM standard for 8" floppy disks. The SA801 also has onboard Large Scale Integration (LSI) integrated circuits which reduce the burden of the disk controller. Examples of functions the on board LSI circuitry provide are FM data/clock pulse separation, and index/sector pulse separation.

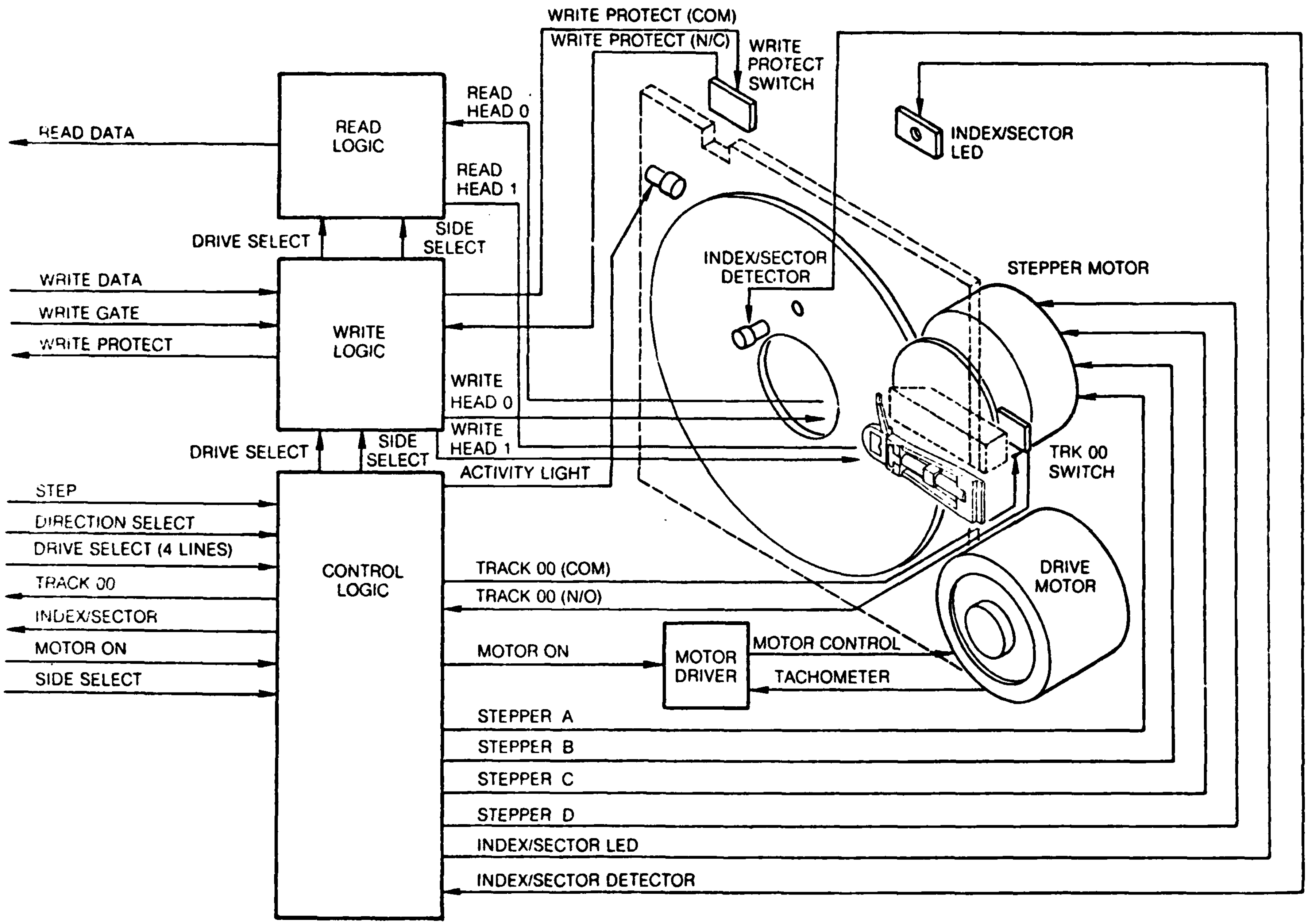


Figure II.4 SA 450 Functional Diagram [Shu C - 81]

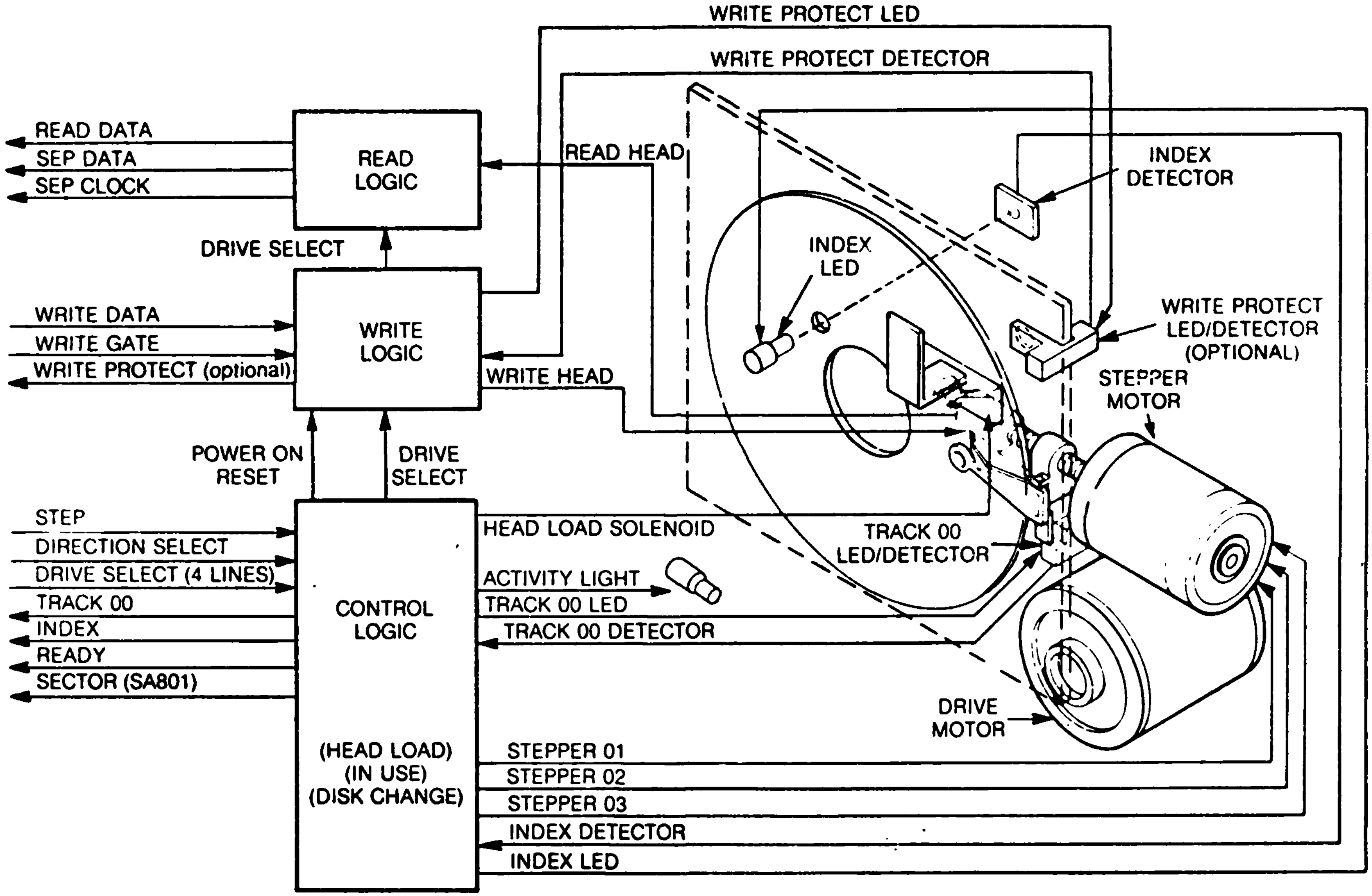


Figure II.5 SA800/801 Functional Diagram [Shu E - 81]

Both the SA400 and SA801 have control electronics to perform the functions listed below.

a) Index/sector hole detection

b) Track zero detection

c) Head position actuator driver

d) Read/Write amplifier detector

e) Write protection

f) Drive ready detection

g) Drive selection [Shu G - 81]

2) SA400 Drive Interface - Figure II.6 depicts all interface connections for the 5 1/4 inch Shugart drives. The interface requirements for each connection are discussed below. Most of these lines have important timing considerations which will be discussed in chapter III.

a) Index/sector - This signal is provided by the drive each time a sector or index hole is detected. The normal state of this signal is high and makes a transition to low for .7 milliseconds when a hole is detected.

b) Drive select 1 to 4 - These are active low connections used to multiplex all control signals to the selected drive. The read/write head will also be loaded when this line is activated. There are jumpers on the drive printed circuit board used to define it as being drive 1, 2, 3, or 4.

c) Motor on - The active state of this line (low) will

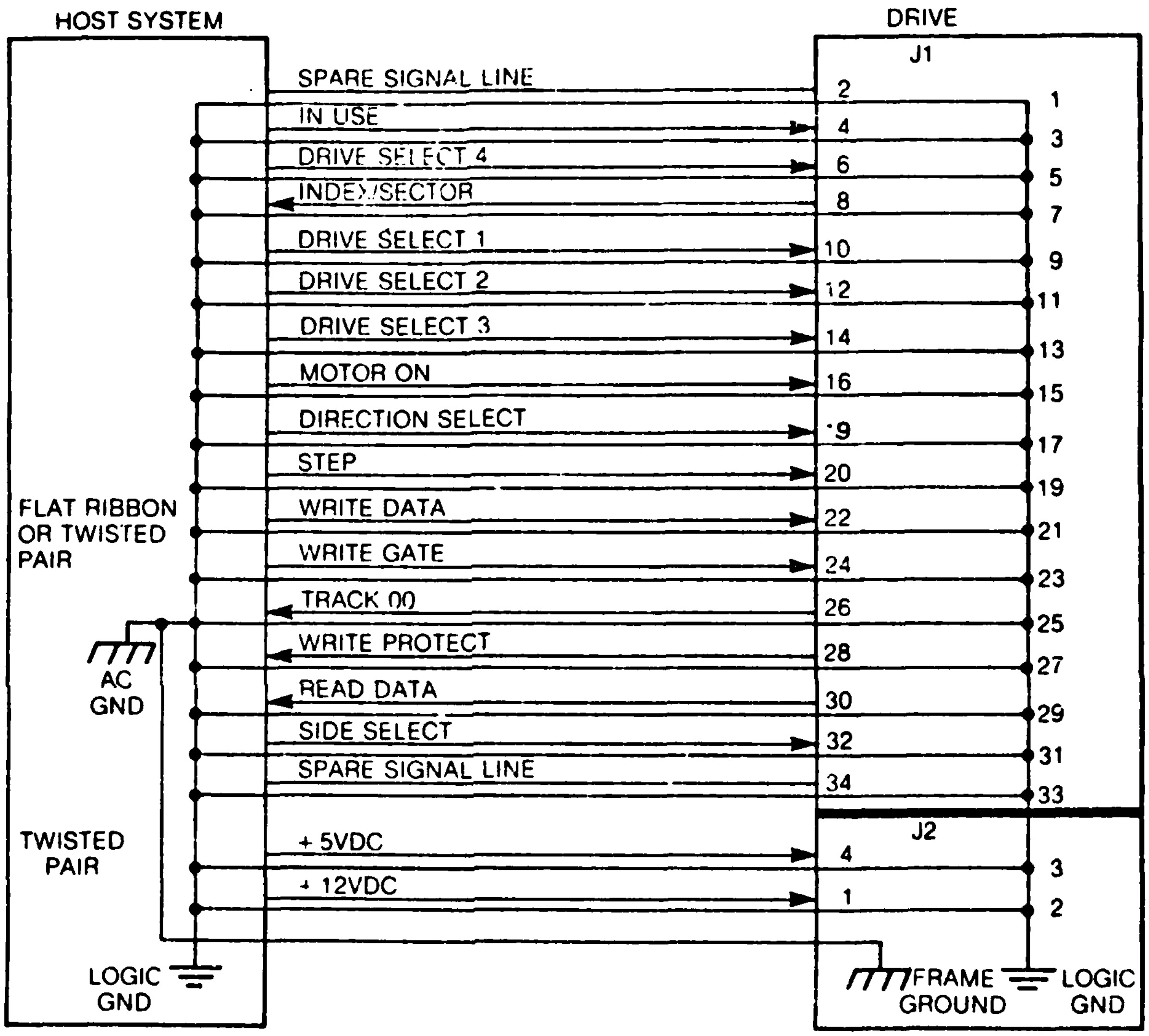


Figure II.6 SA400 Interface Connections [Shu C - 81]

turn on the drive motor. It takes .5 seconds for the motor to come up to speed. A high state on this line will turn the motor off. Head stepping can be accomplished with the motor on or off.

d) Direction select - The state of this line defines the direction the head will move when the step line is activated. If a logical zero is on this line, the head will move toward the center of the disk (in). If an open or logical one is applied to this line, a step pulse will cause the head to move away from the center of the disk (out).

e) Step - This line is normally high. A low pulse will cause the head to move a distance of one track in the direction specified by "direction select".

f) Write data - This line transmits data/clock bits from the host system to the drive to be written on the floppy disk. It is activated only when the write gate is active. Each transition from a high to low will cause a "1" bit to be written to the disk.

g) Write gate - The active (low) state enables the write data circuitry. When this line is high, the read data logic and head stepper logic is enabled.

h) Track 00 - This signal is active (low) only when the head is at the outermost track (track 00).

i) Write protect - This active low signal provides the host system an indication of a write protected

disk. The drive automatically provides write protection with or without this signal.

j) Read data - This line transmits the combined clock and data (RAW data) from the drive to the host system. The normal state of this line is high and transitions briefly to low once for each data/clock pulse. [Shu (C&D) - 81]

3) SA801 Drive Interface - Figure II.7 depicts all interface connections for the 8 inch Shugart drives. The interface requirements for each connection are discussed below. Most of these lines have important timing considerations which will be discussed in chapters 3 and 4.

a) Index - This active low signal is provided once each time the index hole is sensed by the drive. This is true for both hard and soft sectored disks.

b) Sector - This active low signal is provided once for each sector hole detected in a hard sectored disk.

c) Ready - This signal indicates that the disk is installed correctly, the door is closed, and two index holes have been sensed. The Universal Disk Controller does not use this line.

d) Drive Select 1 to 4 - These are active low connections used to multiplex all control signals to the selected drive. The read/write head will also be loaded when this line is activated. There are

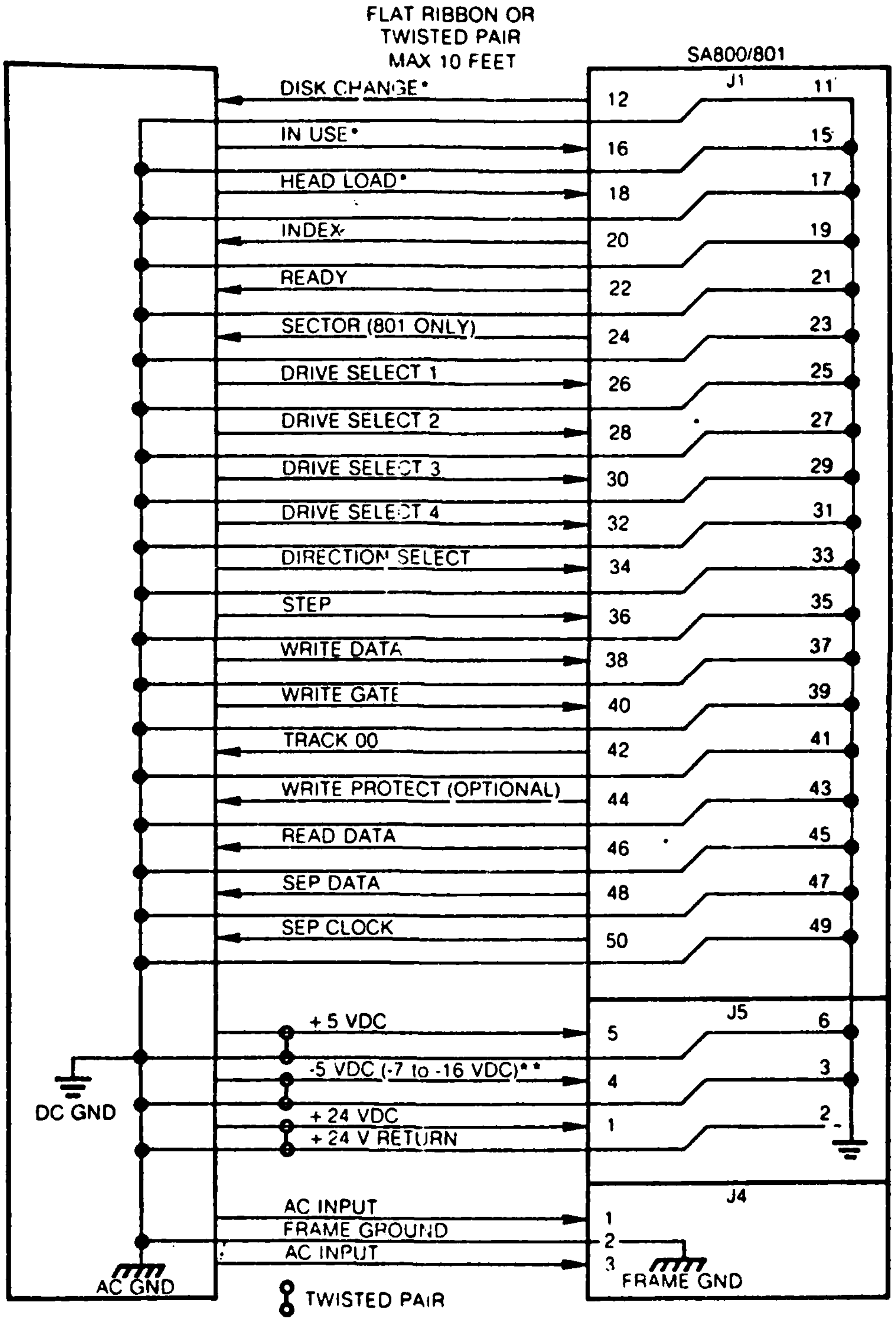


Figure II.7 SA801 Interface Connections [Shu E - 81]

jumpers on the drive printed circuit board used to define it as being drive 1, 2, 3, or 4.

e) Direction select - The state of this line defines the direction the head will move when the step line is activated. A logical zero will cause the head to move toward the center of the disk (in). If an open or logical one is applied to this line, a step pulse will cause the head to move away from the center of the disk (out).

f) Step - This line is normally high. A pulse will cause the head to move a distance of one track in the direction specified by "direction select".

g) Write data - This line transmits data/clock bits from the host system to the drive to be written on the floppy disk. It is activated only when the write gate is active. Each transition from a high to low will cause a "1" bit to be written to the disk.

h) Write gate - When this line is high, the read data logic and head stepper logic is enabled. The active (low) state enables the write data circuitry.

i) Track 00 - This signal is active (low) only when the head is at the outermost track (track 00).

j) Write protect - This active low signal provides the host system an indication of a write protected disk. The drive automatically provides write protection with or without this signal.

k) Read data - This line transmits the combined clock and data pulses (RAW data) from the drive to the host system. The normal state of this line is high and transitions briefly to low once for each data/clock pulse.

l) Separated data - This line provides the separated data from the RAW data for FM encoding only. This line is normally high and transitions to low briefly once for each data pulse.

m) Separated clock - This line provides the separated clock pulse from the RAW data for FM encoding only. This line is normally high and transitions to low briefly once for each clock pulse. [Shu (E&F) - 81]

CP/M File System

The key to understanding how to transfer CP/M files is understanding the CP/M directory. Figure II.8 is an example of a CP/M directory. The directory is logically divided into 32 byte blocks called "File Control Blocks" (FCB). The first byte of each block is the user number (00 - 0F). If the file has been deleted, the first byte will be E5 as shown for FCB 3. Bytes 1 thru 8 are the ASCII bytes (American Standard Code for Information Interchange) for the primary file name. If the file name is less than eight characters then trailing blanks (20 hex) will be used. Bytes 10 thru 12 are the ASCII code for the 3 byte file extension. If the extension is less

than 3 characters then trailing blanks will be used. Byte 13 is the FCB number of one particular file. If a file is large enough to require 3 FCB's, then the first FCB number will be 00, the next 01, and the last 03. Bytes 14 and 15 are not used. Byte 16 is the number of file records (128 byte blocks) the FCB points to. One FCB can point to a maximum of 128 records (one extent). The remaining 16 bytes of the FCB are pointers to the disk sectors and tracks. If less than 16 pointers are needed, then trailing 00's are used. [Hea - 77]

Each pointer represents the exact location of the first logical sector and track for one kilobyte (1024 bytes) of disk data. Each kilobyte of data is referred to as a "group". The directory itself is group 00. If the directory uses two groups, then the first group useable for file storage is group 02. All tracks and sectors prior to the directory are used for the CP/M operating system code. All tracks and sectors after the directory are used for file storage. [Hog - 82]

This chapter has been only a brief analysis of the hardware and software details needed to fully understand the remaining chapters. There are very few good references which give a detailed explanation of floppy disks and floppy disk drives. Some of the best references for this type of information are small manuals published by Shugart. There are many good references for the CP/M operating system. The primary CP/M reference used to write the software for the UDC is the "Osborne CP/M User Guide" written by Thom Hogan.

```
BYTE #: 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
-----------------------------------------------------------------------
FCB 1: 00 4C 38 30 20 20 20 20 20 43 4F 4D 00 00 00 54 .L80     COM....
       02 03 04 05 06 07 08 09 25 26 27 00 00 00 00 00 ................

FCB 2: 00 44 53 4B 20 20 20 20 20 43 4F 4D 00 00 00 1F .DSK     COM....
       0A 0B 0C 0D 00 00 00 00 00 00 00 00 00 00 00 00 ................

FCB 3: E5 57 53 20 20 20 20 20 20 43 4F 4D 00 00 00 7C .WS      COM....
       12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 ................

FCB 4: 00 4D 38 30 20 20 20 20 20 43 4F 4D 00 00 00 80 .M80     COM....
       28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 39 3A 3B ................

FCB 5: 00 4D 38 30 20 20 20 20 20 43 4F 4D 01 00 00 1D .M80     COM....
       3C 3D 3E 3F 00 00 00 00 00 00 00 00 00 00 00 00 ................

FCB 6: 00 57 53 4D 53 47 53 20 20 4F 56 52 00 00 00 80 .WSMSGS  OVR....
       22 23 24 40 41 42 43 44 45 46 47 48 49 4A 4B 4C ................

FCB 7: 00 57 53 4D 53 47 53 20 20 4F 56 52 01 00 00 60 .WSMSGS  OVR....
       4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C ................

FCB 8: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
       E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................

FCB 9: E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
       E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
```

Figure II.8   CP/M Directory

III. Hardware Design

Hardware Overview

The Universal Disk Controller is designed to give the software writter maximum flexibility in choosing a disk format. This criteria makes the hardware design less complex than most disk controllers in use today. The price paid for this lack of complexity is the need for sophisticated software to operate the disk controller. This fact will become apparent in the following chapter (Software Design).

The two primary integrated circuits used in the UDC are the Z80 PIO (parallel input/output chip) and the Z80 SIO (serial input/output chip). Both of these chips are already contained on the SQ Single Board computer which greatly simplifies the hardware design. Many of the interface lines run directly between the PIO and disk drive.

There are two types of interface lines required for the disk drives. These are control lines and signal lines. The control lines have the job of selecting drives, turning the motor on and off, and stepping the read/write head. The signal lines are used to transfer the index pulse, sector pulse, read data, and write data.

The overall layout of the disk controller is depicted in figure III.1. Most of the control lines are handled thru channel A of the Z80 PIO. Channel B of the PIO is used for all signal lines except data transfer which is the job of the SIO.

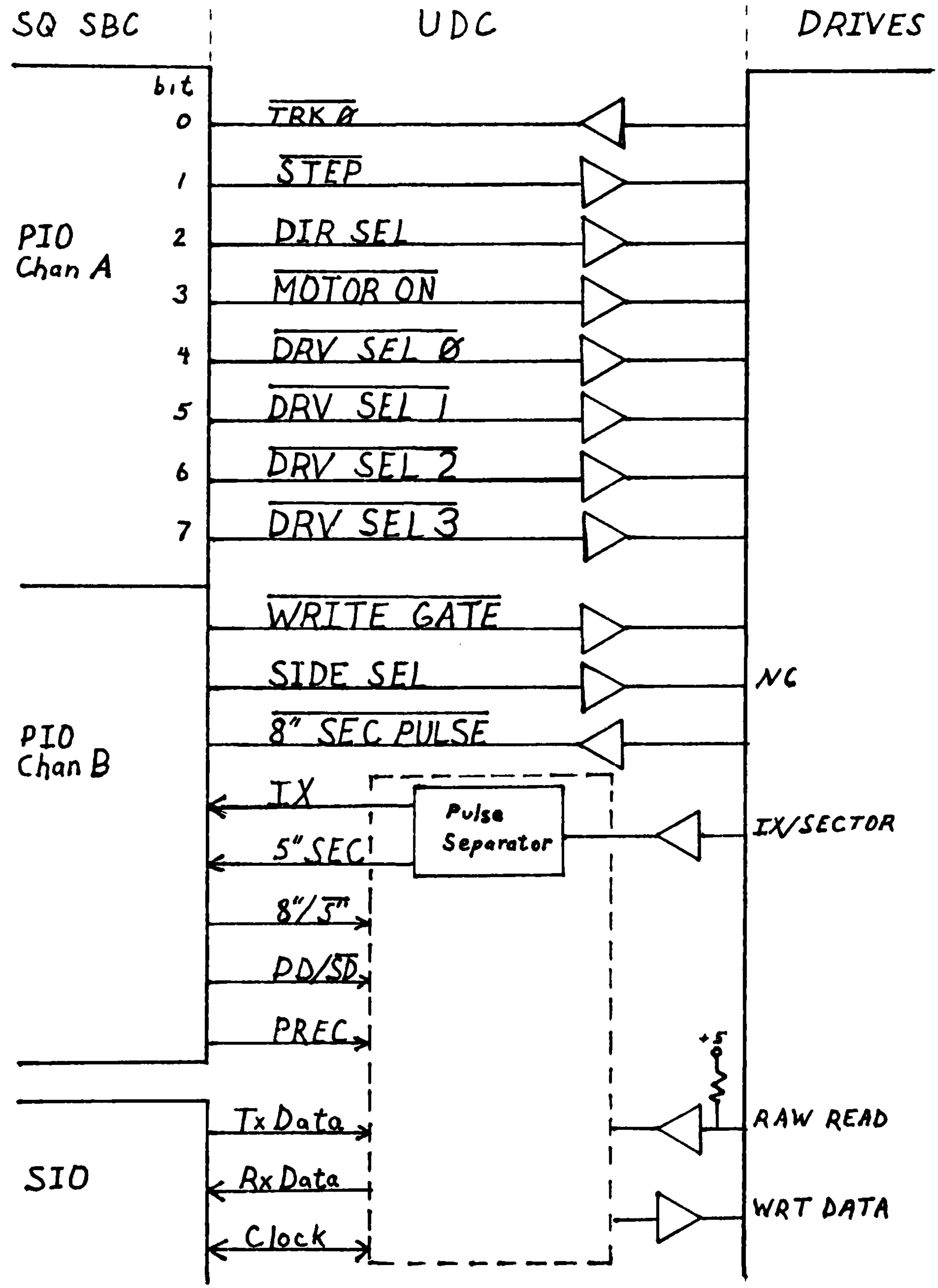


Figure III.1 UDC Interface

Two Western Digital LSI integrated circuits were used to simplify the job of read and write data pulse timing. These chips are the WD1691 and the WD2143-03. The WD1691 is called a "Floppy Support Logic Chip" and the WD2143 is a "Four Phase Clock Generator".

The WD1691 was designed to minimize external logic required by Disk Controllers. This integrated circuit is used as the primary component in read data synchronization. Also, it is used in conjunction with the WD2143 Four Phase Clock Generator to condition data for write precompensation. These chips are compatible with both 5 inch and 8 inch drives. [WD data book]

All other integrated circuits used in the Universal Disk Controller are small scale integration such as gates, buffers, and monopulse generators.

The discussion of hardware design is divided into three categories. These are miscellaneous circuits, data recovery circuit, and write circuit. The miscellaneous circuits will be discussed first because they generate signals used by the primary data transfer circuits.

Miscellaneous Circuits

1) Index/Sector Pulse Separator - An index/sector pulse separator is required only by the 5 inch drives. As discussed in chapter II, the Shugart 8 inch drives generate their own separated index and sector pulse.

Figure III.2 depicts a 10 sectored hard sector disk. When rotated at 300 rpm, the index pulse will be sensed 300 times per minute or once every 200 milliseconds. The sector holes will be sensed 10 times as often or once every 20 milliseconds. The same logic can be used to derive the timing for a 16 sectored hard disk.

The timing diagram III.4 shows the various outputs of the separator depicted in figure II.3. Signal A is the original drive signal after being buffered by a 74244 non-inverting bus driver. Signal B is the output of the first 555 monopulse generator. This signal is used to synchronize with the sector pulses. A pulse is generated each time the drive signal transitions from high to low. It lasts for 12.0 milliseconds. This value was chosen as a compromise in timing requirements between 10 and 16 sectored disks. This pulse is not triggered by an index pulse because it has not yet recovered to its normal high state when the index pulse arrives.

The second 555 generates a mask pulse that is used to separate the sector and index pulses. It can be seen in figure III.4 that C NOR A is an active high index pulse. C' NOR A is an active high sector pulse.

The value of the capacitors and resistors used with the 555's were estimated using the following formula.

pulse duration = 1.1 X RC [Lan - 74]

The values attained from this formula were fine tuned with a

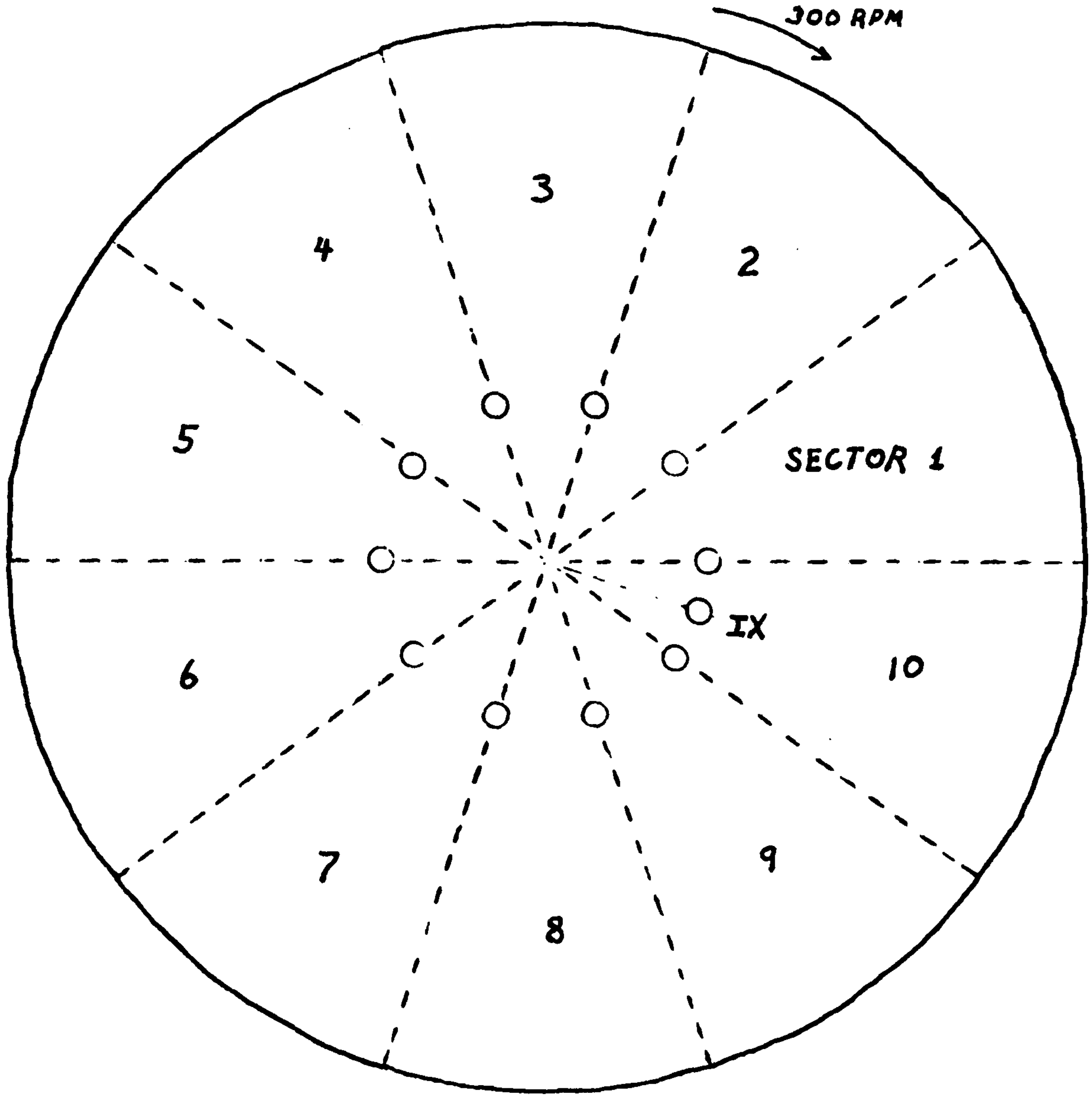


Figure III.2   10 Sectored Hard Sectored Disk

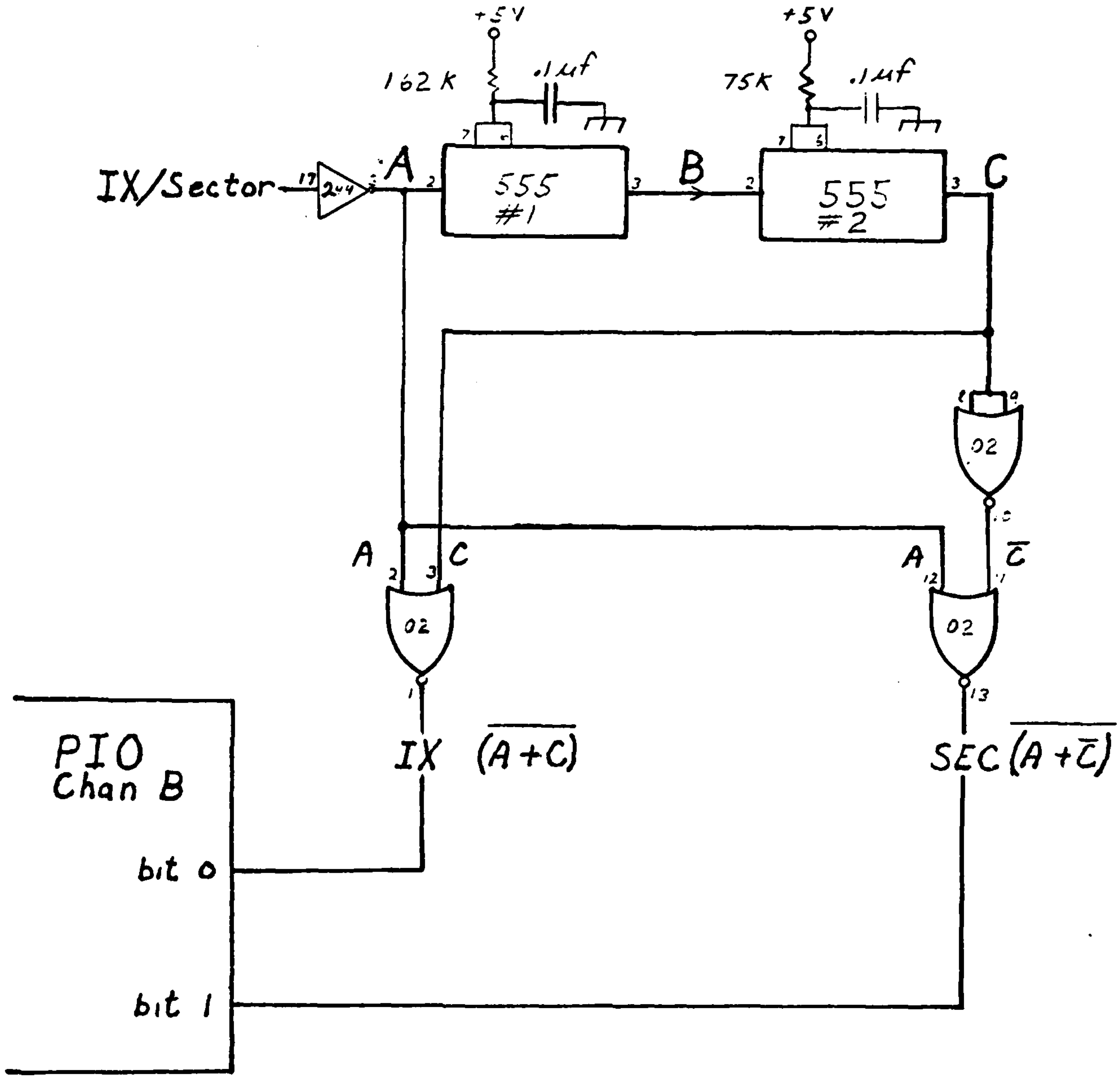


Figure III.3 IX/Sector Pulse Separator

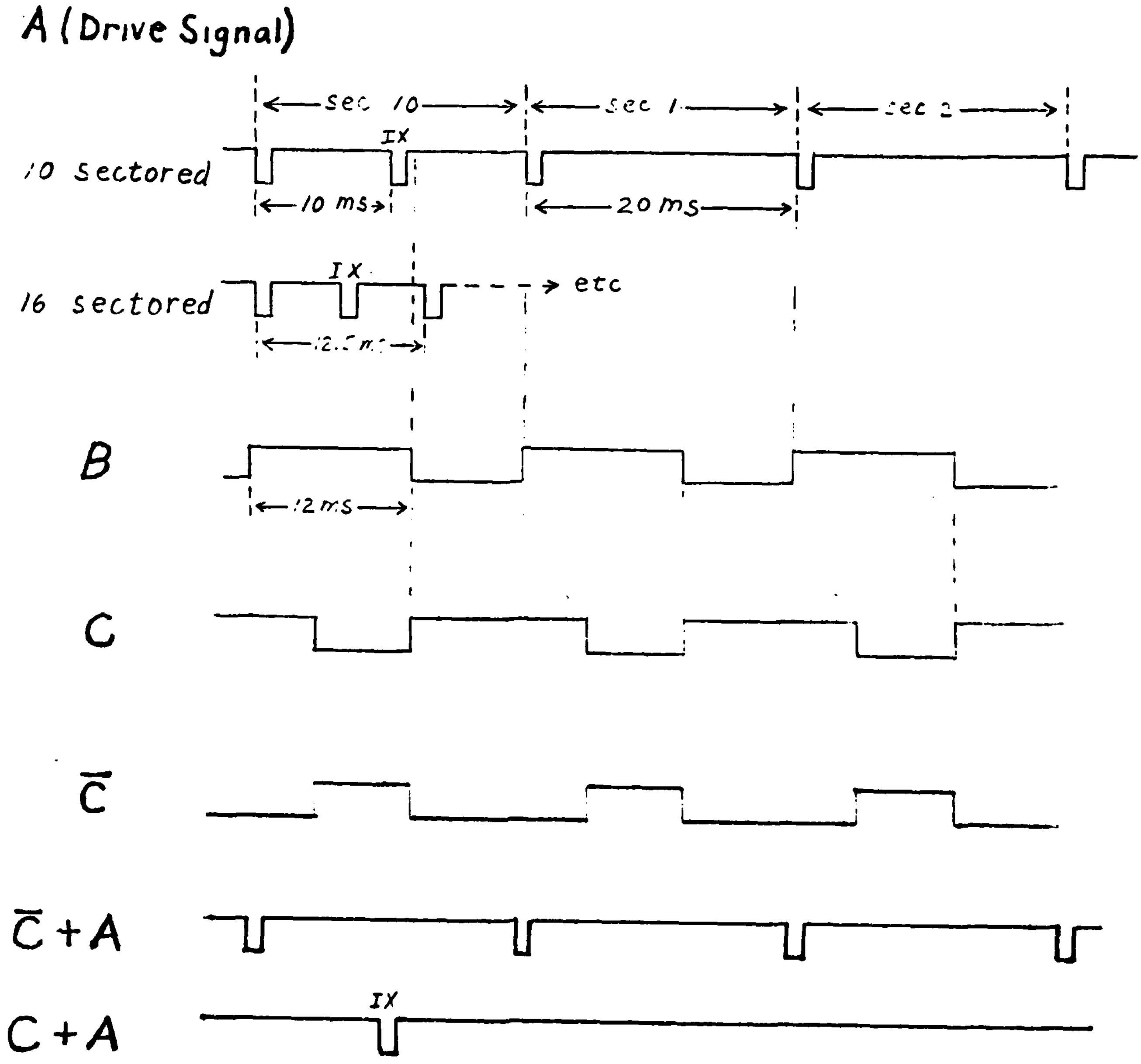


Figure III.4 IX/Sector Timing

rheostat and multimeter. The component values shown in figure III.3 are used to attain pulse lengths depicted in the timing diagram III.4.

2) Decoder - The UDC's decoder (figure III.5) is used to attain an individual signal for 5" single density, 5" double density, and 8" single density disks. These outputs are derived from bits 3 and 4 of the PIO channel B. Bit 3 is high for 8" and low for 5". Bit 4 is high for double density and low for single density. The outputs of the decoder are used as inputs to several circuits that will be discussed later.

3) Crystal Oscillator - This circuit (figure III.6) is used to generate stable frequencies for write operations. A 4 MHz crystal is used in conjunction with an MC4024 to derive a 4 MHz +/- 1% square wave. This signal is divided by a 4 bit counter (8291). The two most significant outputs of the counter are 500 KHz and 250 KHz. 500 KHz is needed to write 5" double density and 8" single density. 250 KHz is needed to write 5" single density. 1 MHz would be required to write 8" double density; however this is not used because the Z80 SIO in conjunction with a 4 MHz CPU has a maximum clock frequency of 700 KHz.

4) Clock Multiplexer - This circuit (figure III.7) serves two purposes. First, it decides which clock is connected to the SIO; either the transmit clock or receive clock. Second, it selects the transmit clock speed; either 500 KHz or 250 KHz.

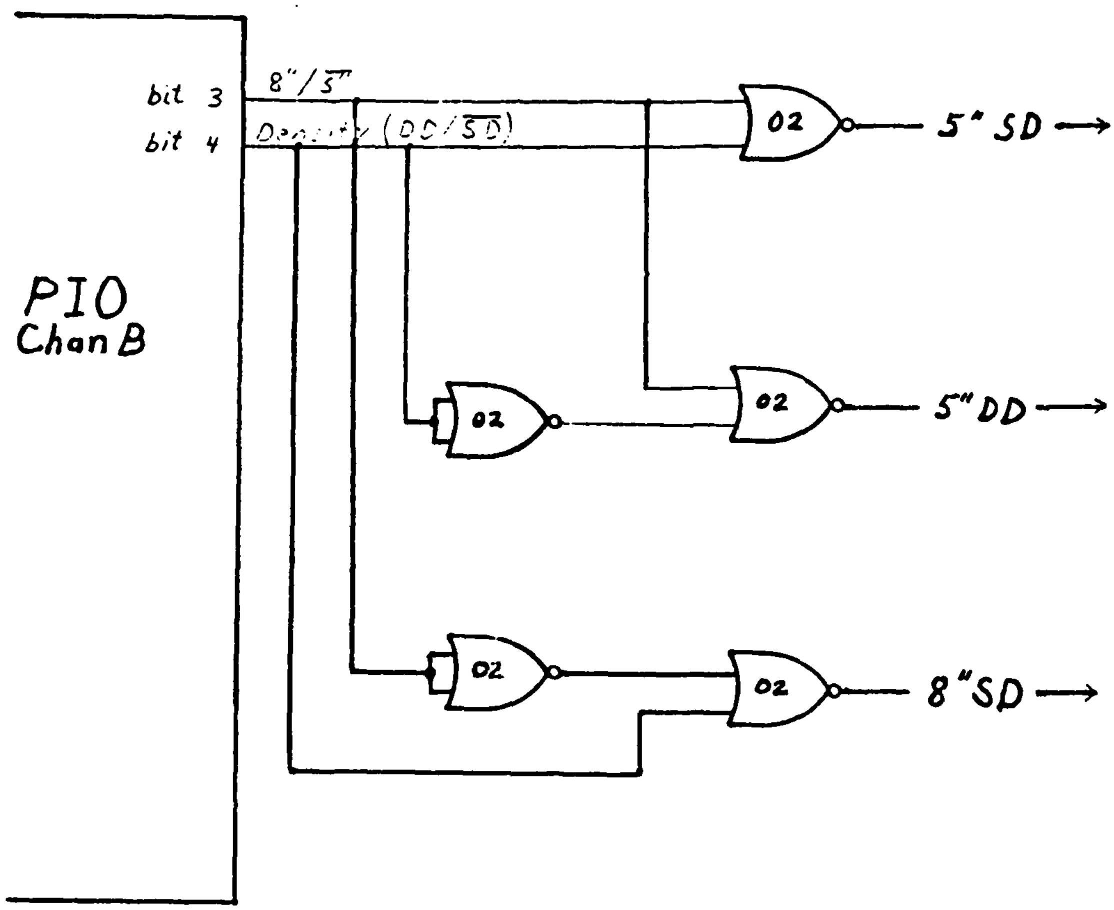


Figure III.5 Size/Density Decoder

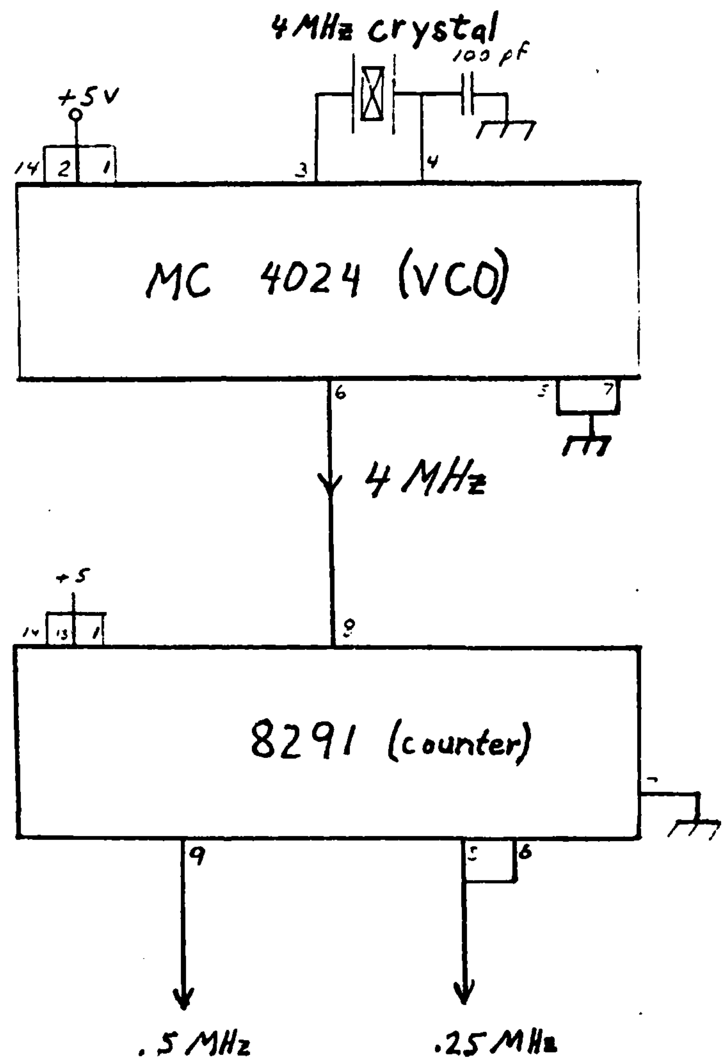


Figure III.6 Tx Clock

The integrated circuit used is a dual 4 line to 1 line multiplexer (74153). Table III.1 is a truth table for this chip. MUX A has the transmit and receive clocks as inputs, while MUX B has the two clock speeds as inputs. Table III.2 is derived from the MUX truth table. This shows that the receive clock connects to both C1 and C3 (MUX A), the transmit clock to both C0 and C2 (MUX A), 500 KHz to C0 and C1 (MUX B), and 250 KHz to C2 and C3 (MUX B). The output of MUX A connects directly to the SIO clock pin while the output of MUX B connects to several circuits needing the transmit clock frequency.

Data Reccvery Circuit

1) Overview - A block diagram of the read circuit is given in figure III.8. The tasks of this portion of the UDC are as follows.

a) Establish synchronization with RAW read pulses.

b) Put clock pulses in form usable by the SIO.

c) Put data pulses in a form usable by the SIO.

2) Timing - Figure III.9 depicts an example of RAW data as it appears from the disk drive for 5" single density. The distance between the clock pulses is 8 microseconds and the distance between a clock pulse and a data pulse is 4 microseconds. Since the UDC reads both the data and clock pulses, an SIO clock frequency of (1 / 4 us) or 250 KHz is

SELECT INPUTS		DATA INPUTS				Out
B	A	Co	C1	C2	C3	
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

Table III.1 MUX Function Table

Fast .5 MHz Slow .25 MHz	SELECT B Slow/Fast	SELECT A RX/TX	Output follows:
fast Tx	0	0	Co
fast Rx	0	1	C1
Slow Tx	1	0	C2
Slow Rx	1	1	C3

Ch A: $RX = C_1 + C_3$
$TX = C_0 + C_2$

Ch B: $fast = C_0 + C_1$
$slow = C_2 + C_3$

Table III.2 Implementation Table

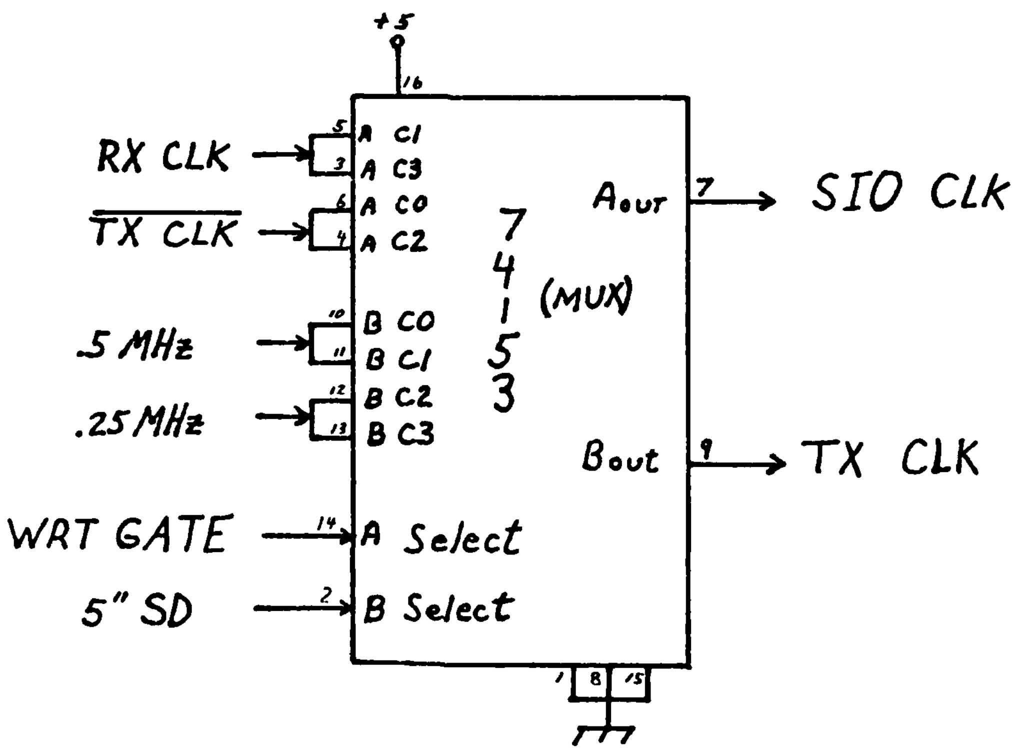


Figure III.7 Multiplexer

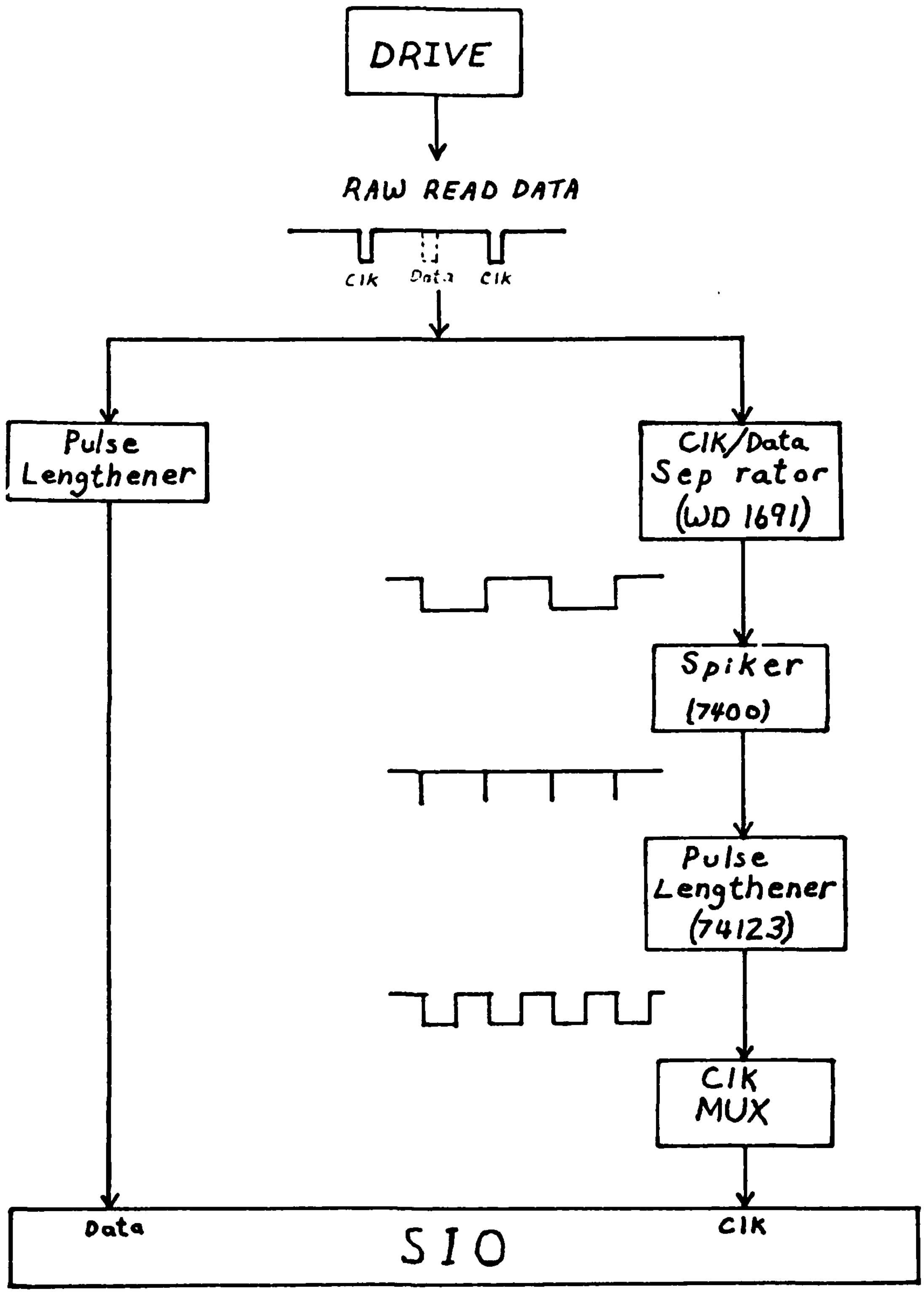


Figure III.8 Read Circuit Block Diagram

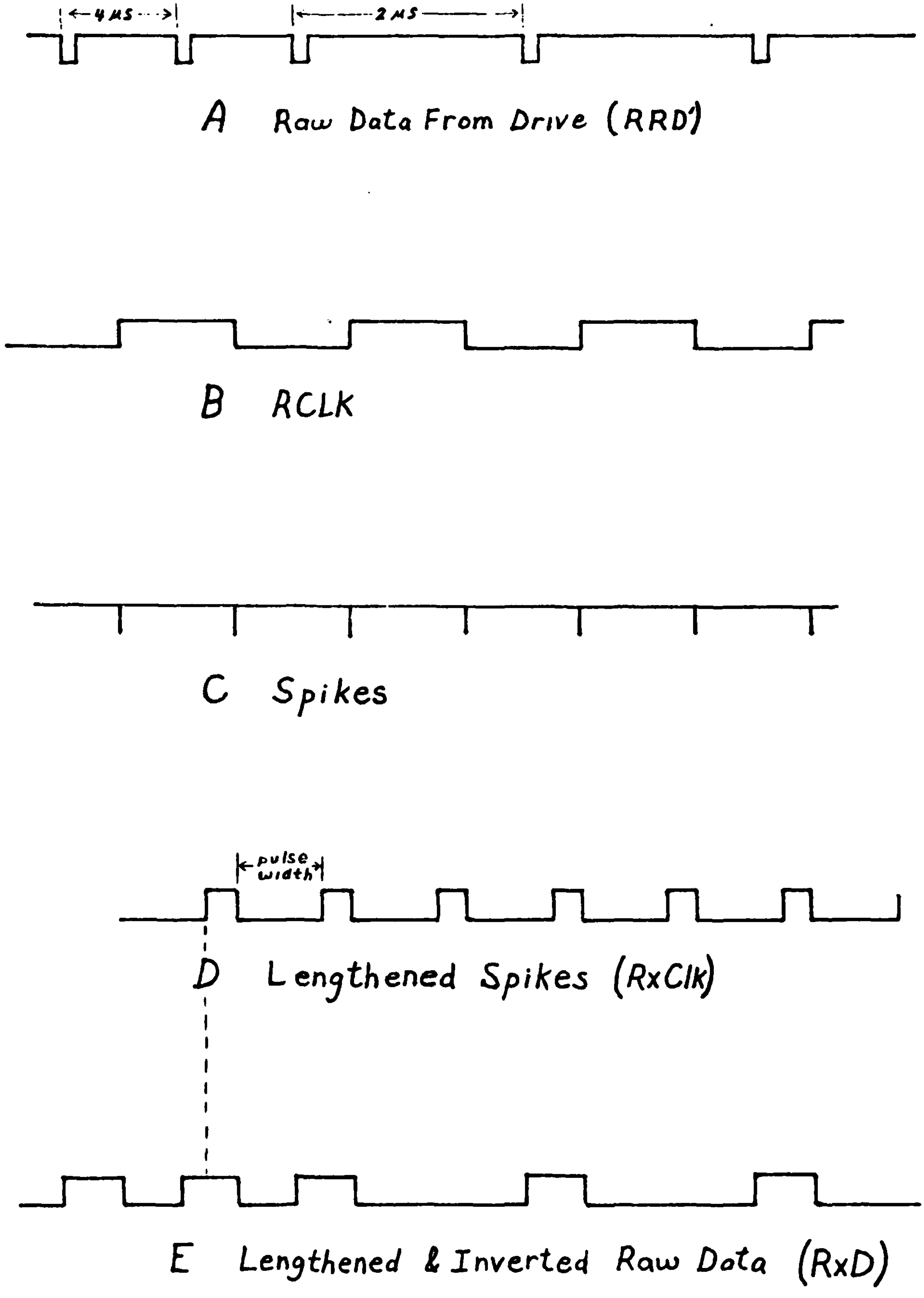


Figure III.9 Read Timing

needed for 5" single density. A frequency twice this is needed for 5" double density and 8" single density (500 KHz).

3) Synchronization - The job of read synchronization is accomplished by the WD1691 in conjunction wich a 74S124 (VCO). As already mentioned, the WD1691 "Floppy Support Logic" handles conditioning for both reading and writing. Only that portion of the chip pertaining to data recovery will be depicted in this section's diagrams.

The WD1691 RCLK separator (figure III.10) has four inputs: RRD', X2, WG, and VFOE/WF. RRD' is the RAW read signal from the disk drive. X2 is used to double the output clock frequency. WG (write gate) is used to select the data recovery circuit when low and the write circuit when high. VFOE/WF is a signal needed by the WD1795 disk controller chip. The UDC just uses it as another select for read/write operations, therefore it is tied with the WG pin. [Wes - 82]

The WD1691 has three outputs: PU, PD', and RCLK. PU and PD' (pull up and pull down) are used to control the VCO output frequency. RCLK (read clock) is the final output of the WD1691 which is a square wave synchronized to RRD'.

When WG and VFOE are low, the read circuit is active. PU goes high when an increase in VCO frequency is needed to lock to RRD', else it is at a hi-Z state (open circuit). PD' goes low when a decrease in VCO frequency is needed, else it is at a hi-Z state. By tying PU and PD' together a signal is

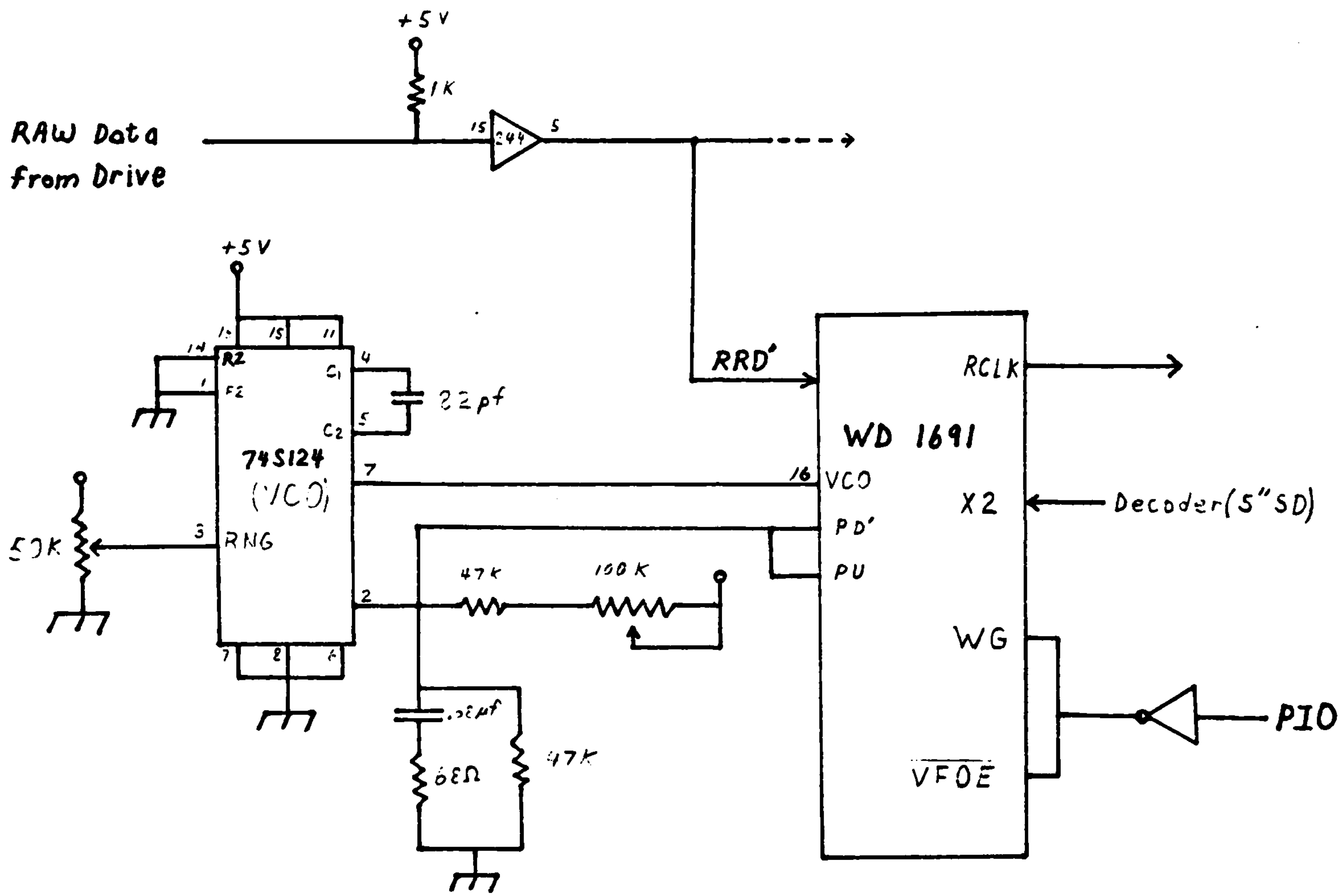


Figure III.10 RCLK Separator

created which goes high requesting an increase in frequency, low requesting a decrease in frequency, and will remain hi-Z indicating no adjustment is necessary.

The requirements outlined in the Western Digital Components Handbook specify that the VCO needs to have a nominal output frequency of 2 MHz with an input voltage of 2.4V. To achieve the 2.4 volts, the 100K rheostat (figure III.10) is adjusted. To acquire 2 MHz, the 50K rheostat is adjusted. All component values depicted in figure III.10 were taken directly from the 1983 Western Digital Components Handbook.

The final output of the WD1691 (RCLK) is a divide-by-16 of the VCO frequency when X2 is high, and a divide-by-8 when X2 is low. The 5" SD output of the decoder is tied to the X2 pin. Therefore, RCLK is 125 KHz when 5" single density is selected, and 250 KHz when 5" double density or 8" single density is selected. This is half the frequency required by the SIO to read both the clock <u>and</u> data bits sent from the drive. The reason for this apparent error is that the WD1691 was designed to recover only the data pulses (not data and clock pulses).

4) Receive Clock - This signal is derived directly from RCLK. RCLK is an input to the NAND spiker (figure III.11). The output of this circuit is a low spike for each positive <u>and</u> negative edge of RCLK (see figure III.9B). The frequency of these spikes is twice the frequency of RCLK which is the frequency needed by the SIO receive clock.

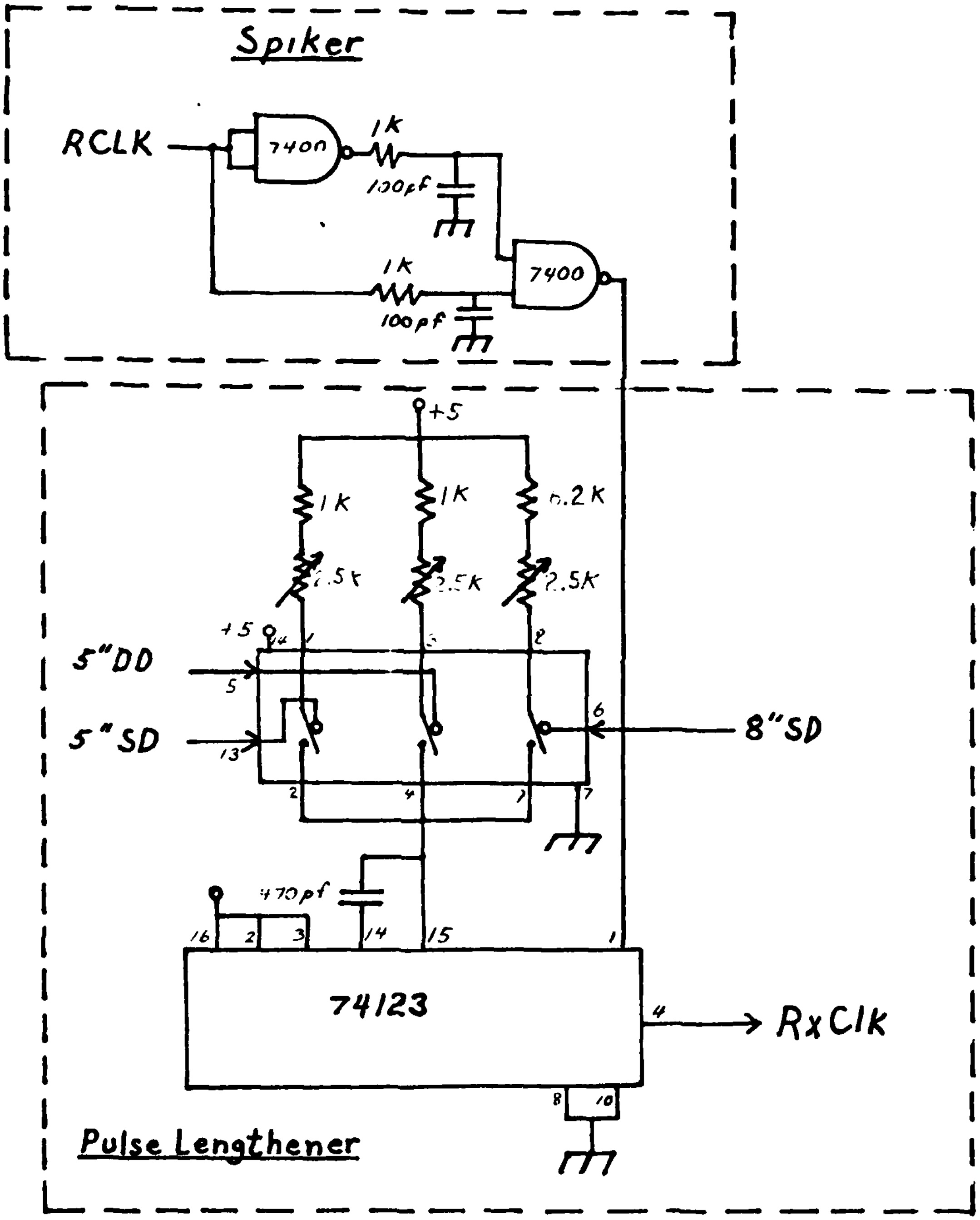


Figure III.11 RxClk Circuit

The next and final step is to lengthen the spikes (see figure III.9D) using a 74123 monopulse vibrator (figure III.11). This signal is the receive clock used by the SIO.

The SIO takes data on the rising edge of the receive clock. To put the rising edge of the receive clock where it is needed, the pulse length is adjusted. This adjustment is accomplished by the 4016 CMOS analog switch, rheostats, and monopulse generator depicted in figure III.11. The pulse length is directly proportional to the resistance value attached to pin 15 of the monopulse generator.

The CMOS IC is used to switch in three discrete values of resistance. These resistances give pulse lengths required by 5" SD, 5" DD, and 8" SD. The three rheostats are used to adjust the individual resistances.

5) Receive Data - The only changes made to RRD' to acquire SIO data pulses are depicted in figure III.9E. RRD' is lengthened and inverted using a 74123 monopulse multivibrator.

6) Result - The inputs to the SIO now look like those depicted in figure III.9D and III.9E. The SIO will sample RxD on the rising edge of RxClk.

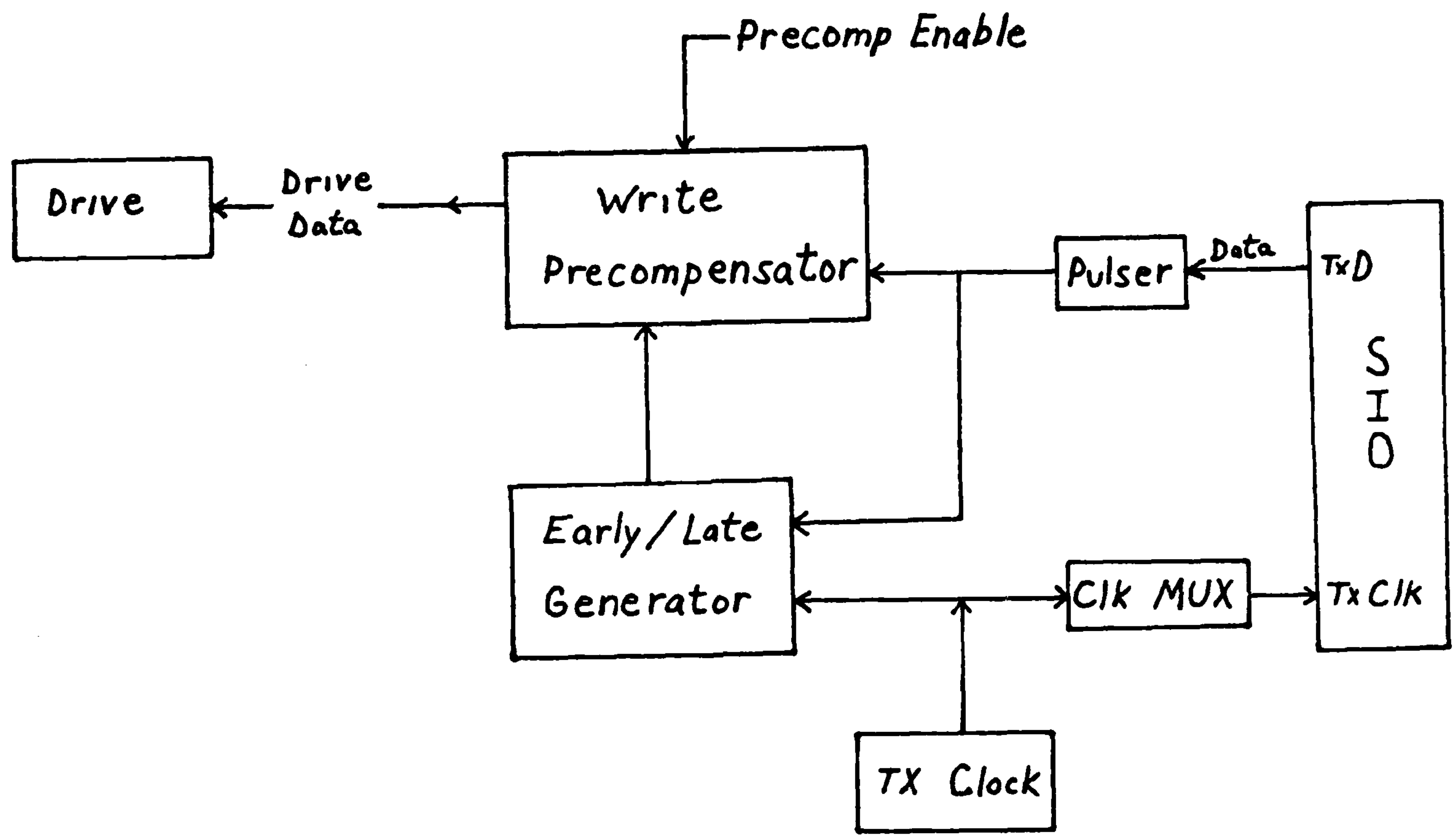


Figure III.12   Write Circuit Block Diagram

Write Circuit

1) Overview - A block diagram of the write circuit is depicted in figure III.12. The tasks of this portion of the UDC are as follows.

a) Convert the SIO output data stream to a signal useable by the disk drive.

b) Precompensate the data stream if necessary.

2) SIO data output - Figure III.13 depicts the timing characteristics of the Z80 SIO output data. The clock input to the SIO originates from the crystal oscillator. The frequency will be 250 KHz for 5" SD and 500 KHz for 5" DD and 8" SD. The transmit data changes on the falling edge of the clock.

The first step in conditioning the SIO data is to convert each "1" data bit into a 300 nanosecond pulse (disk drive requires a 200 to 2100 nanosecond pulse). This is accomplished by a 74123 monopulse multivibrator (figure III.14) and 7408 AND gate. The input to the monopulse generator is an inverted clock. The clock is inverted because the rising edge of the the clock occurs in the center of the data bit, but the 74123 triggers on a falling edge. When the output of the monopulse generator is AND'ed with the data, the result is an active high stream of data pulses usable by the WD1691.

3) Write precompensation - As discussed in chapter II,

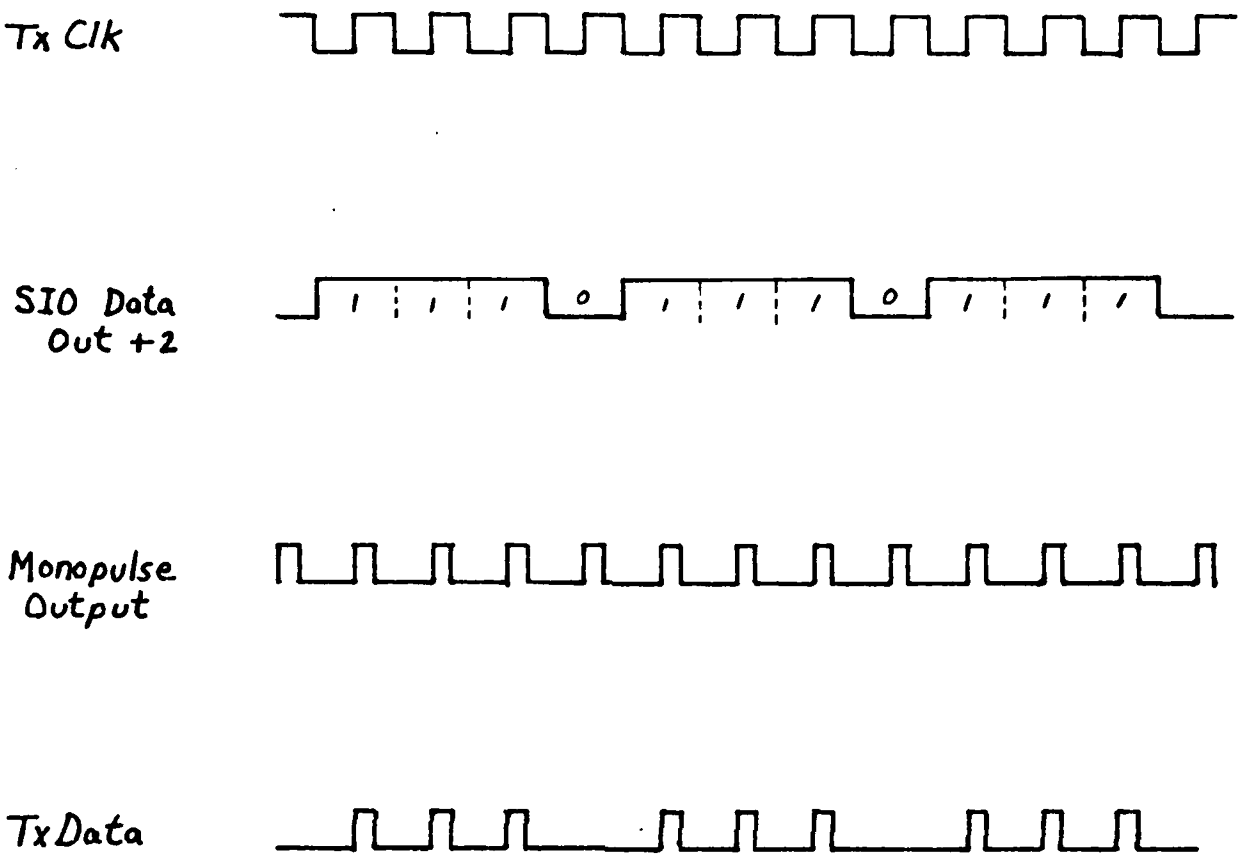


Figure III.13 Write Timing Diagram

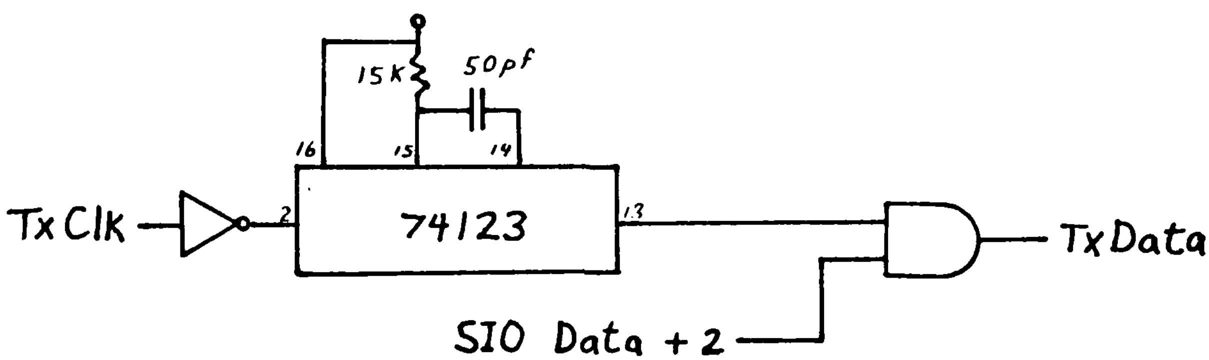


Figure III.14 Tx Data Pulser

write precompensation is necessary at times when bit shift is significant enough to increase the possibility of write errors. Write precompensation is handled by the WD1691 Floppy Support IC (figure III.15), WD2143 Four Phase Clock Generator, and an early/late signal generator made from small scale integration (SSI) chips.

Write precompensation is selected when PCE (precomp enable) is high and X2 (pin 15) is low. PCE is connected to PIO channel B, bit 5. This gives the UDC software selectable precompensation.

When precompensation is disabled, WDIN (write data in) appears at WDOUT (write data out) unchanged. When precomp is enabled, the signals early and late (E and L) are used to select a phase input (phase 1 to 4) on the leading edge of WDIN. The STB (strobe) line goes high when this occurs, causing the WD2143 to start its pulse generation. The early write pulse occurs at phase 1, nominal write pulse at phase 2, and late pulse at phase 3. Phase 4 is used to reset the STB line.

Figure III.16B depicts an example of a data stream used to explain the design of the early/late signal generator. An early signal is needed when a "1" occurs after another "1" and before a "0". A late signal is needed when a "1" occurs after a "0" and before a "1".

Figure III.17 depicts a state diagram of a machine that accomplishes the requirements layed out in the above paragraph. State A occurs any time the previous bit was a "0".

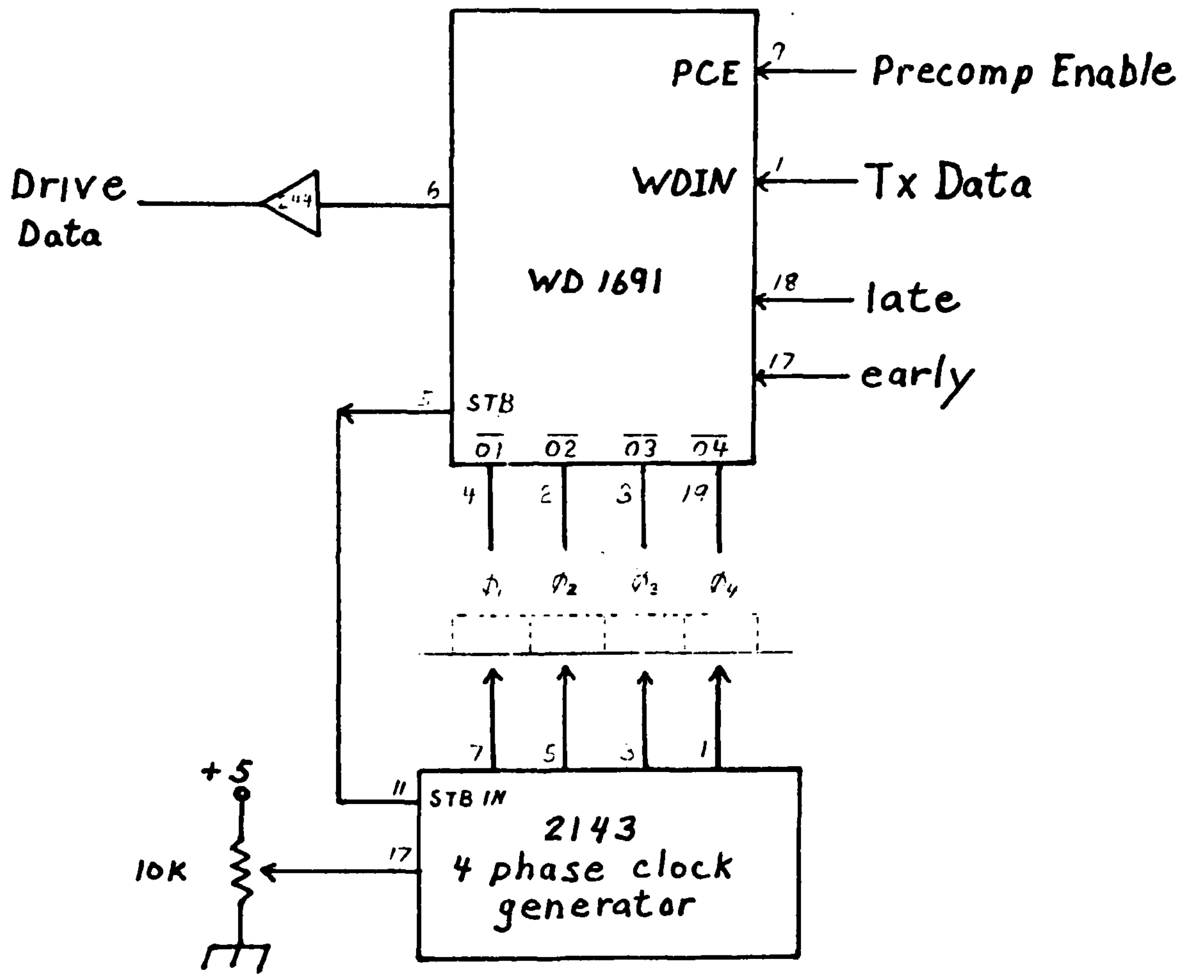


Figure III.15   Precompensation Circuit

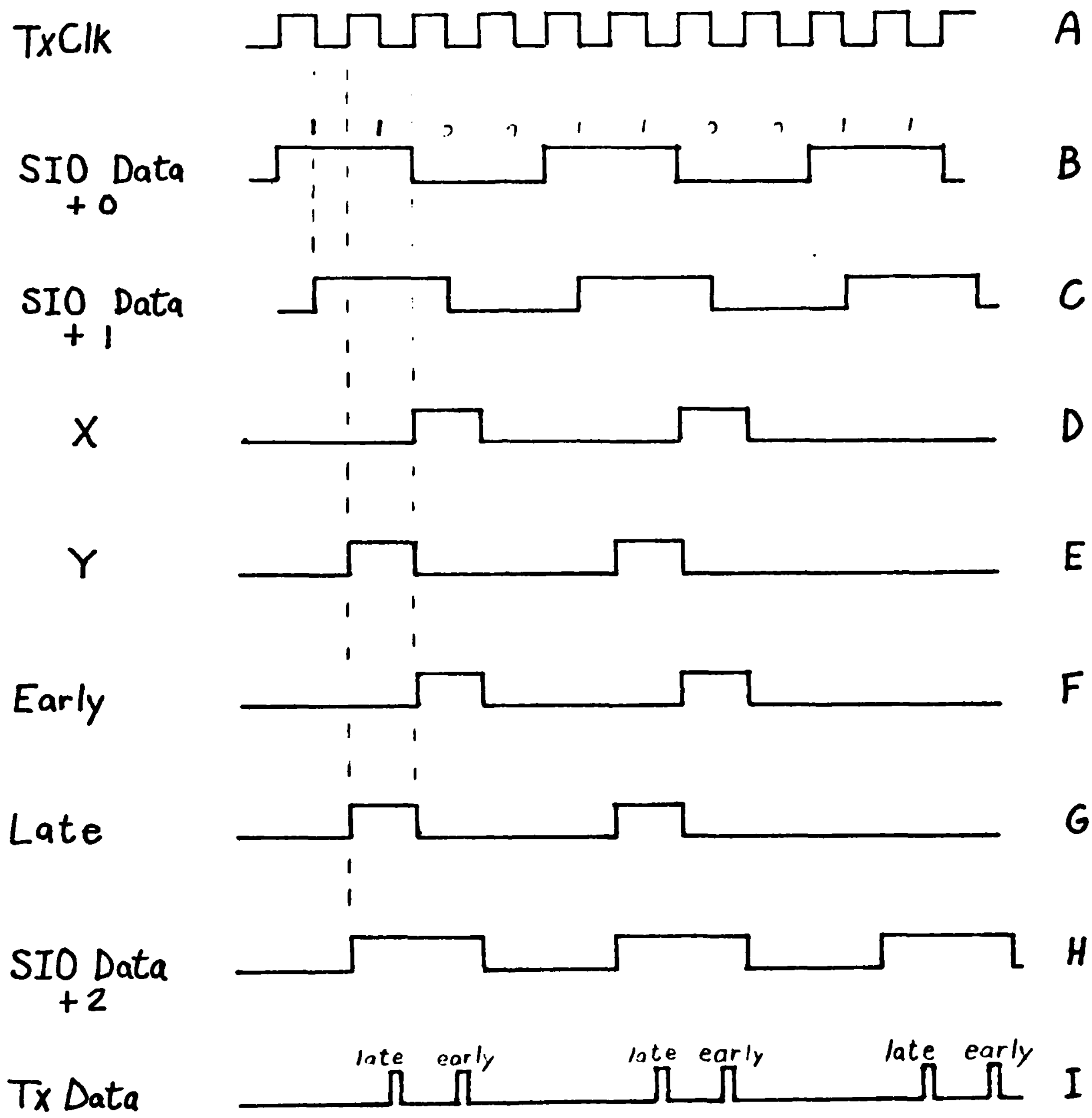


Figure III.16 Early/Late Timing

State B occurs whenever the previous two bits were "01". State C occurs whenever the previous two bits were "11". Transitions between A and B never result in an early or late signal. A late pulse is generated when a transition is made from B to C. An early pulse is generated when transition is made from C to A.

This finite state machine was implemented with 7474 D-type flip-flops. The state table is shown in table III.3 The three states require 2 flip-flops which leave one undefined state. Instead of leaving the D inputs to the undefined states "don't care's", they were forced to logic "0". This allows the machine to transition to the start state if the undefined state occurs due to noise or other undesirable conditions. The late output of the last undefined state was forced to a logic "1". This allowed a reduction which resulted in a requirement for one less integrated circuit. The reduced boolean equations and the resulting hardware design are depicted in figures III.18 and III.19.

The timing diagram III.16 shows a requirement for three phases of the data pulses: "SIO Data + 0", "SIO Data + 1", "SIO Data + 2". These phases were acquired by a shift register constructed from 7474 D-type flip-flops shown in figure III.20. The data input to this shift register is "SIO Data + 0", the output of the first stage is "SIO Data + 1", and the final output is "SIO Data + 2". The X and Y pulses of figure III.16 were derived from "SIO Data + 1". E and L

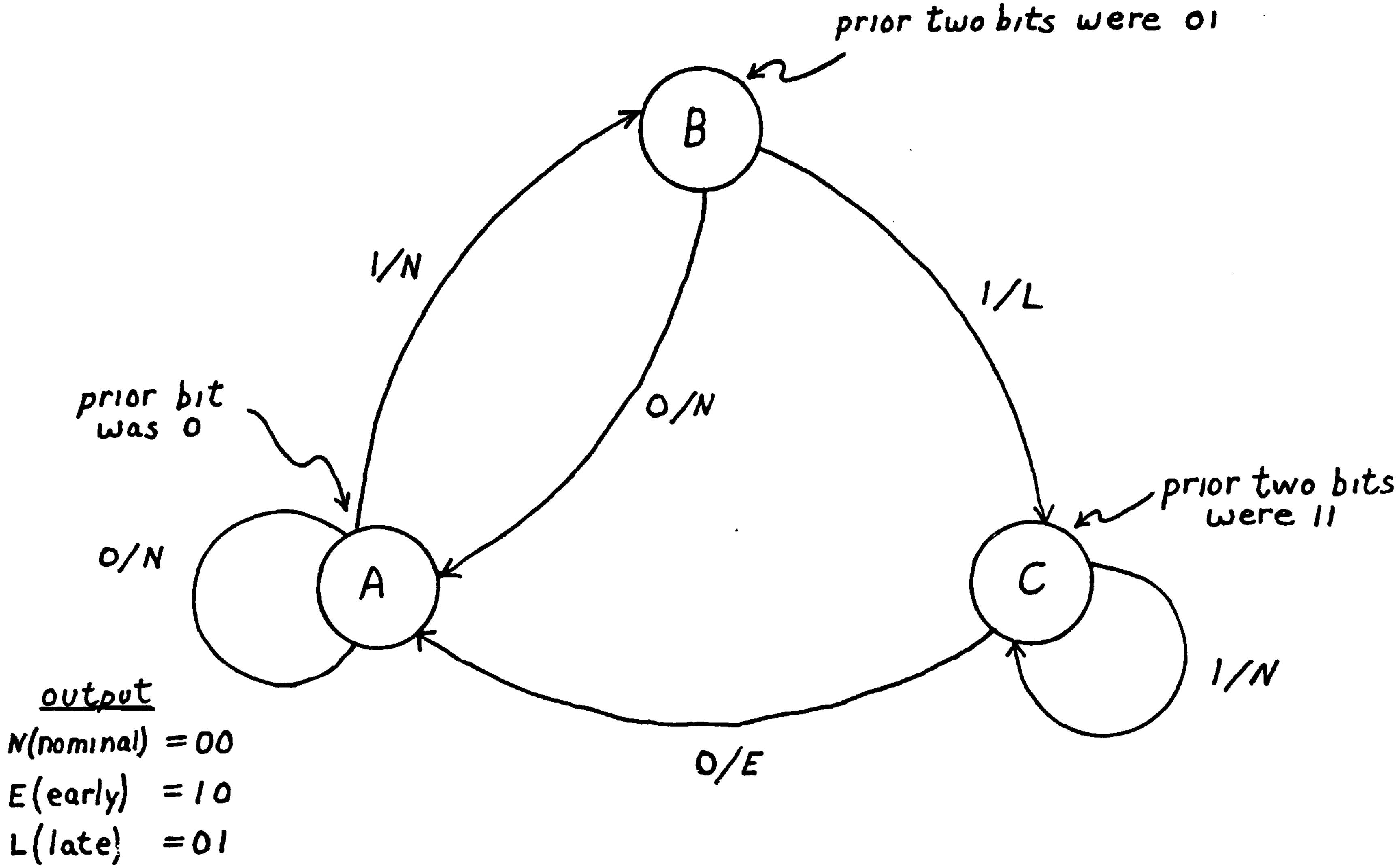
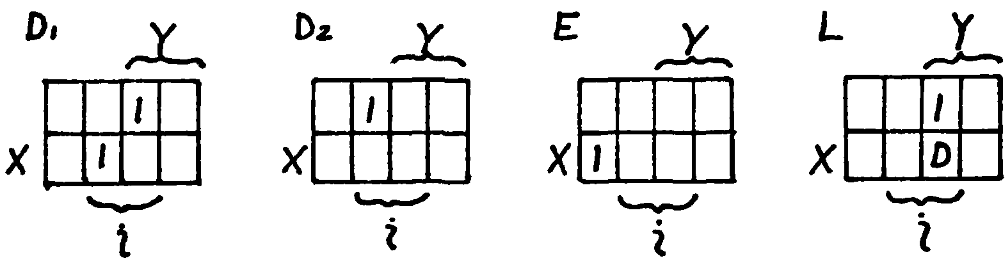


Figure III.17 Early/Late State Diagram

Present State			Input	Next State			Output		F-F input	
	X	Y	i		X_{i+1}	Y_{i+1}	E	L	D_1	D_2
A	0	0	0	A	0	0	0	0	0	0
	0	0	1	B	0	1	0	0	0	1
B	0	1	0	A	0	0	0	0	0	0
	0	1	1	C	1	0	0	1	1	0
C	1	0	0	A	0	0	1	0	0	0
	1	0	1	C	1	0	0	0	1	0
U	1	1	0	A	0	0	0	0	0	0
	1	1	1	A	0	0	0	1	0	0

Table III.3 State Table



$$D_1 = \bar{X} Y i + X \bar{Y} i = \overline{(X+\bar{Y}+\bar{i})(\bar{X}+Y+\bar{i})}$$

$$D_2 = \bar{X}\bar{Y} i = \overline{(X+Y+\bar{i})}$$

$$E = X\bar{Y}\bar{i} = \overline{(\bar{X}+Y+i)}$$

$$L = Y i$$

Figure III.18 Boolean Reduction

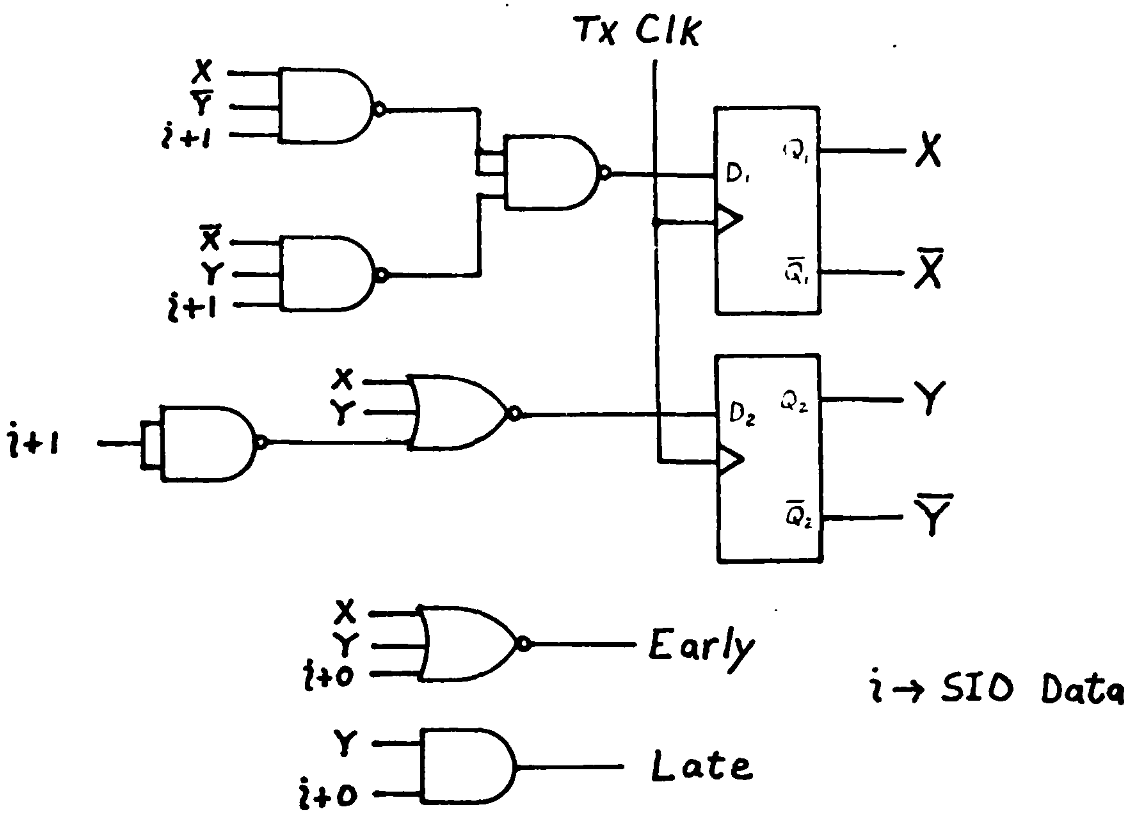


Figure III.19 Early/Late Generator

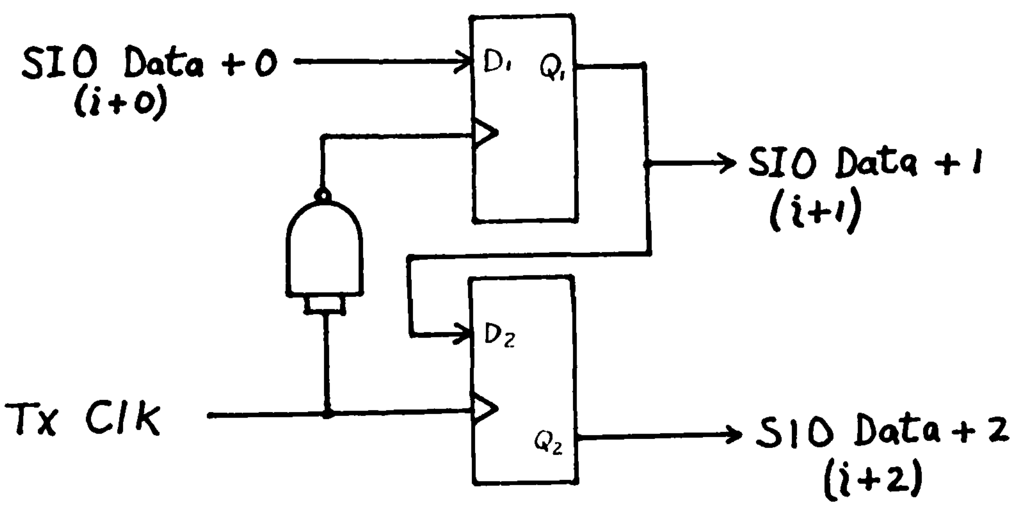


Figure III.20 Data Shifter

were derived from "SIO Data + 0". "SIO Data + 2" is used as the input to the WD1691 because these data pulses fall in the center of the corresponding early and late pulses.

The final precompensated data pulses are depicted in figure III.16I. If precompensation is enabled, the data output pulse of the WD1691 will occur late if L is high at the time the data pulse enters. The reverse is true if E is high.

All "data pulses" mentioned in this chapter are actually the combination of data and synchronization pulses. The final extraction of the actual data is the responsibity of the UDC's software which will presented in the next chapter.

IV. Software Design

Software Overview

The software designed to operate the Universal Disk Controller is divided into three parts as depicted in figure IV.1. The high level box in this figure is written in the "C" programming language and is responsible for controlling all file manipulation. Its file name is "TRANSIT.C". The middle boxes represent routines written in Z80 assembly language. These routines read and write contiguous sectors to the various disk systems. File names of the current systems in use are "IBM", "NEC", and "NS" (Northstar). The lowest level box contains many library routines designed to operate the UDC. The library routines are written in Z80 Assembly Language and use the file name: "UDCLIB".

The software design will be presented in a bottom up manner beginning first with the UDC library routines. Once an understanding is achieved of how to use the library routines, various systems can be added with minimum difficulty.

All source code is listed in the appendix.

Library Routines

The discussion of the UDC library routines will be divided into 6 sections: PIO Programming, SIO Programming, Drive Head Control, Hard Sector Modules, Soft Sector

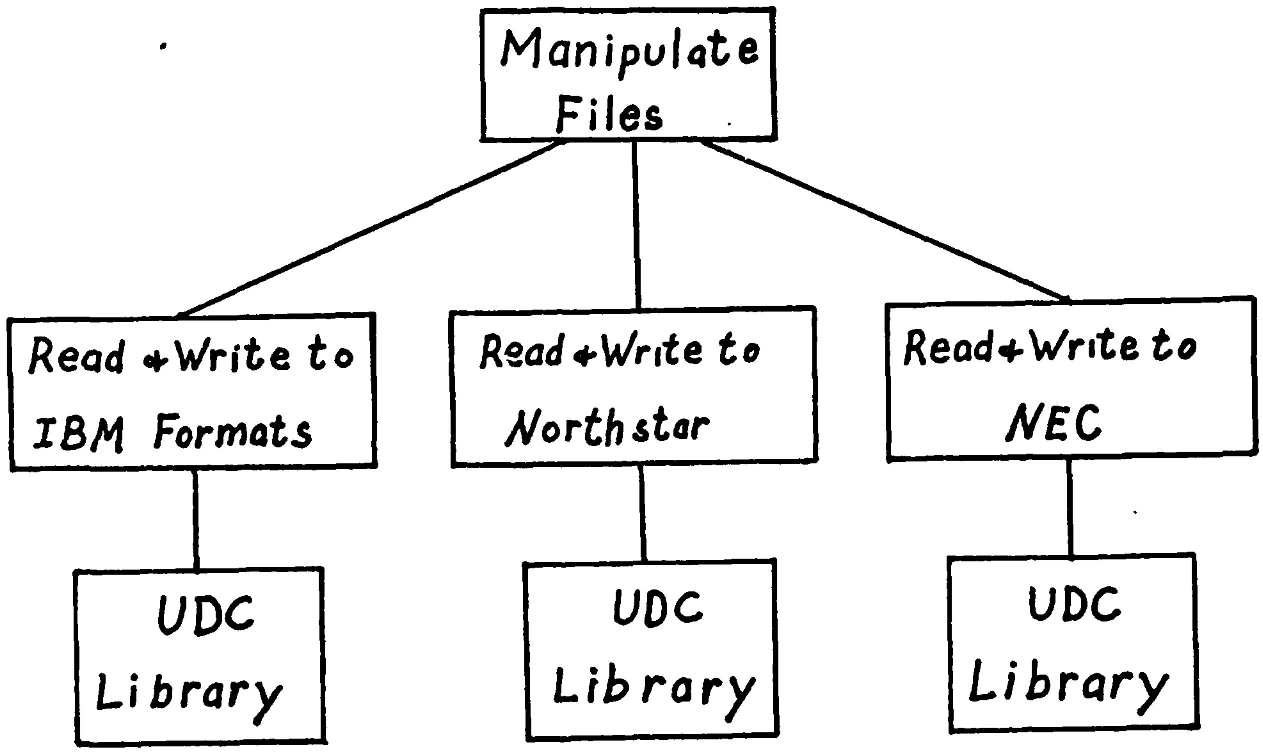


Figure IV.1 Software Structure

Modules, and CRC modules. All sector transfer modules use either FM or MFM encoding. No other encoding schemes are used within this library. The following is a list of the more important library routines and a quick definition of there purpose.

a) PIOINIT - Initialize PIO.

b) SIOINIT - Initialize SIO.

c) DRVSET - Select desired drive, size, and density.

d) DSELECT - Deselect all drives.

e) HOME - Move drive head to track zero.

f) SEEK - Move head to desired track.

g) IXPLS - Wait for index pulse to arrive.

h) HIGHP - Wait for sector pulse to go active.

i) FNDHSEC - Wait for desired hard sector to arrive.

j) FNDFSEC - Wait for desired soft FM sector to arrive.

k) FNDMSEC - Wait for desired soft MFM sector to arrive.

l) RXENABL - Enable SIO receiver.

m) TXENABL - Enable SIO transmitter.

n) RECEIV - Receive disk data and discard clock bits (for FM <u>or</u> MFM encoded data).

o) RDHDFM - Read one hard sectored, FM encoded block of data.

p) RDHDMFM - Read one hard sectored MFM encoded sector.

q) RDSFM - Read one soft sectored, FM encoded block of data.

r) RDSMFM - Read one soft sectored, MFM encoded block of data.

s) WRHDFM - Write one hard sectored, FM encoded block of data.

t) WRHDMFM - Write one hard sectored MFM encoded sector.

u) WRSFM - Write one soft sectored, FM encoded block of data.

v) WRSMFM - Write one soft sectored, MFM encoded block of data.

w) TXFER - Transfer an unlimited number of contiguous sectors using one of the above sector transfer routines.

x) CMPCRC - Compute cyclic redundancy value using the polynomial: x^16 + x^12 + x^5 + 1.

y) CRCCHK - Check CRC using the above polynomial.

1) PIO Programming - Only the PIO programming considerations relevant to the UDC library routines will be discussed in this section. A full explanation of Z80 PIO programming can be found in the "ZILOG DATA BOOK".

The PIO used by the UDC is an integral part of the SQ single board computer. It has two channels which will be referred to as channel A and B. Each channel has eight bits which can be used in a variety of modes. The UDC uses only mode 3 which is the "bit I/O" mode. This mode allows the user to individually define and use each bit as input or output. The purpose of each bit for both channels is listed below. The bits with asterisks are direct representations of the corresponding drive interface pin outlined in chapter 2.

Channel A

0* - Track 0 (active low)

1* - Step (active low)

2* - Direction select (low = in, high = out)

3* - Motor on (active low)

4* - Drive select 1 (active low)

5* - Drive select 2 (active low)

6* - Drive select 3 (active low)

7* - Drive select 4 (active low)

Channel B

0 - Index pulse (active high)

1 - Sector pulse for 5" drives (active high)

2* - Write gate (active low)

3 - Size select (high = 8") (low = 5")

4 - Density select (high = DD) (low = SD)

5 - Precompensation enable (active high)

6* - Side select (high = side 0) (low = Side 1) (not usable on SA400 and SA801)

7* - Sector pulse for 8" drives (active low)

Module (PIOINIT) - This is a short module which selects mode 3 in both channels of the PIO, then defines each bit as being input or output. Once the bits are defined, the write gate is deactivated, all drives are deselected, and the drive motors are turned on.

Module (DRVSET) - This module is called just prior to any disk accessing. The desired drive, density, and size are set. The desired size and density is passed by the "A" register. Bit 3 is set for 8 inch and reset for 5 inch drives. Bit 4 is set for double density and reset for single

density. All other bits must be zero. The desired drive is passed by the memory location using the symbol: "DRIVE". A delay of two revolutions of the disk will occur within this routine. This delay insures the drive motors are up to speed prior to a read or write operation.

2) SIO Programming - Only the SIO programming considerations relevant to the UDC library routines will be discussed in this section. A full explanation of Z80 SIO programming can be found in the "Zilog Data Book".

The SIO used by the UDC is an integral part of the SQ single board computer. It has two channels, however only one is used by the UDC (channel B). The following is a list of the primary parameters needed when programming the SIO for use with the UDC.

a) no CRC checking (by SIO)
b) no interrupts
c) 8 bits per character / no parity
d) synchronous mode
e) X1 clock
f) 16 bit synch character
g) synch char load inhibit [Zil - 82]

Module (SIOINIT) - This module resets the SIO and selects the synchronous mode. The 16 bit synch character will be set to the contents of the DE register pair. This module is called every time a transmit or receive operation

is about to begin or a new synch character is needed.

Module (RXENABL) - "Receive Enable" enables the SIO receive mode, then enables the SIO wait state generator.

MODULE (TXENABLE) - "Transmit Enable" enables the SIO transmit mode, then enables the SIO wait state generator.

Module (STOP) - This routine turns off the wait state generator.

The character transmission and reception routines of the library all use wait states. This is done to meet the needs of the high data transfer rates and still have enough time to encode or decode the data/clock bits as they are transferred.

It is not necessary to accomplish the encoding or decoding as the data is transferred, but this does save time and a lot of memory space. If decoding were to be accomplished after one or more sectors were read, exactly twice the amount of memory would be required because both the data and clock bits would have to be stored as they were read. If encoding were to be accomplished prior to writing one or more sectors, again exactly twice the amount of memory would be required to hold the encoded data (data and clock bits).

Experimental analysis has shown that on the average 240 CPU clock cycles pass for each byte transferred with a 2 MHz SIO clock (5" single density), and 120 CPU clock cycles pass with a 4MHz SIO clock (5" double density or 8" single density). This means that about 30 one byte Z80 machine instructions can be accomplished between "IN" or "OUT"

instructions at the 2 MHz rate, and about 15 one byte instructions at the 4 MHz rate. This number of instructions is more than adequate to decode or encode FM encoded data. The algorithm to encode using the MFM format is very difficult to implement with this limited number of machine instructions, but is possible.

The Z80 SIO transmits and receives the least significant bit of a byte first. This conflicts with the way most single chip disk controllers record data on a disk (most significant bit first). Therefore, decoding and encoding of data is a two step process. Clock bits must be inserted or extracted, then the byte must be flipped. The following example is used to explain data decoding.

FM EXAMPLE: *2 bytes received by SIO:* 11010111 11111101

1) Flip the bytes: 11101011 10111111

2) Discard the clock bits to make one byte of data. Every other bit is a clock bit starting with the first bit:

   c1c0c0c1 c0c1c1c1 or 1001 0111

   Result: D7 FD decodes to 97 (hex)

MFM decoding uses exactly the same steps. This is true because in the above example it does not matter whether or not a clock bit is a "1" or a "0". It will be discarded regardless. FM encoding uses the same process in reverse. MFM encoding uses a similar process but has added steps used to comply with the rule: clock bit is "1" only if surrounding data bits are both zero.

3) Drive Head Control Modules - These modules control all read/write head movement. There are important timing constraints to be met and are depicted in figure IV.2. All of these times are met with a high degree of accuracy using a software delay loop. This module ("DELAY") uses the BC register pair as a counter. A delay of .1 milliseconds times the BC value will occur when this module is called.

Each drive has its own track register which is updated every time the head is moved. These registers use the symbol names: "TRKRG0" to "TRKRG3" corresponding to drive 0 to drive 3.

Module (STPOUT) - Moves the head out one track each time it is called.

Module (STPIN) - Moves the head in one track each time it is called.

Module (HOME) - Repeatedly calls "STPOUT" until bit 0 of PIO channel A goes to zero (Track 00 indication). The corresponding track register is set to zero.

Module (SEEK) - Repeatedly calls "STPOUT" or "STPIN" as necessary to move the head to the desired track. The corresponding track register is updated to match the new head position. The desired head position is passed to this module using the memory location with symbol name: "TRACK".

4) Hard Sector Modules - All UDC read and write hard sector routines assume the following disk format is used for each sector.

1) 16 bytes of synch characters (zero's)

2) one data address mark (usually FB)

3) unlimited number of data bytes

4) two CRC bytes

Prior to calling a hard sector transfer module, the head must be positioned within the boundaries of the sector _prior_ to the desired sector. This is necessary to allow contiguous sector reads. The first instruction in all these modules waits for the sector pulse to go active prior to executing its read or write operation.

The first time a sector is transferred on one particular track, the head can be positioned within the bounds of the previous sector by calling "SECWAIT". This routine waits for the index pulse, then counts off the sector pulses until the sector prior to the desired sector is reached. The remaining sectors can be read or written simply by continually calling the sector transfer module without calling "SECWAIT". Contiguous sectors can be read in this manner without waiting for an extra disk revolution.

The following global memory locations are used to pass parameters to the hard sector transfer routines.

a) "DTA" - address of first byte of data

b) "BYTSEC" - bytes per sector

Prior to calling a hard sector write routine, the CRC value must be computed and the "IX" register must point to the first CRC byte. If reading a sector, the disk's CRC

value will be placed in the memory location pointed by the "IX" register.

5) Soft Sector Modules - Soft sectored disks do not have physical holes in the diskette marking the sector boundaries. Instead, sector boundaries are determined by reading identification fields recorded on the disk. This method has the advantage of flexibility and reliability over hard sectoring, and the disadvantage of complexity.

Although "flexibility" was called an "advantage" in the above paragraph, it could easily be called a "disadvantage" by the makers of universal disk controllers. The number of various soft sector formats available to a manufacturer is unlimited. Therefore, creating a system to read all formats is quite a challenge. There are however, two highly popular disk formats used: IBM 3740 and IBM System 34. Both of these formats are implemented in the UDC library.

The names of the UDC sector transfer modules which implement these formats are as follows.

a) RDSFM - read one IBM 3740 sector (FM)

b) WRSFM - write one IBM 3740 sector (FM)

c) RDSMFM - read one IBM System 34 sector (MFM)

d) WRSMFM - write one IBM System 34 sector (MFM)

These modules should be called with the following parameters set.

a) sector number in register "E"

b) track number in memory location: "TRACK"

c) disk transfer address in memory location: "DTA"

d) bytes per sector in memory location: "BYTSEC"

e) "IX" register pointing to precomputed CRC value if writing

f) "IX" register pointing to memory location to receive disk CRC value if reading

g) skew information (discussed in later paragraph)

IBM 3740 is a single density FM format. Each sector consists of the following bytes.

a) 6 synch bytes (zeros)

b) ID address mark (FE with C7 clock pattern)

c) track number

d) side number (00 or 01)

e) sector number

f) sector length flag (00)

g) 2 ID field CRC bytes

h) 11 synch bytes (FF or 00, normally 00)

i) 6 synch bytes (00)

j) data address mark (FB)

k) data

l) 2 CRC bytes

The ID address mark and data address mark have missing clock bits (C7 clock pattern). This technique is used to make the address marks unique from other information recorded on the disk. These address marks are perfect to use as 16 bit synch characters within the SIO for two reasons: 1) they will not appear in data fields, 2) the bytes

directly following the address mark are the needed information.

CRC bytes are used for both the ID field and the data field. The polynomial used to compute the CRC value is $X^{16} + X^{12} + X^5 + 1$. The bytes included in the ID CRC computation begin with the ID address mark and end with the sector length flag. The bytes included in the data CRC computation begin with the data address mark and end with the last data byte.

The IBM System 34 is a double density (MFM) format. Each sector consists of the following bytes.

a) 12 synch bytes (00)

b) 3 index address marks (A1 with 0A clock pattern)

c) 1 ID address mark (FE)

d) track number

e) side number (00 or 01)

f) sector number

g) sector length flag (01)

h) 2 ID field CRC bytes

i) 22 fill bytes (4E)

j) 12 synch bytes (00)

k) 3 index address marks (A1)

l) 1 data address mark (FB)

m) data

n) 2 data CRC bytes

o) 54 fill bytes (4E)

The "A1" index address marks are made unique from all other encoded information on the disk by excluding the last clock bit ("0A" clock pattern). This unique mark greatly simplifies the task of recognizing sector boundaries.

The CRC value is computed using the same polynomial as the IBM 3740 format. The bytes included in the ID CRC computation begin with the first index address mark and end with the sector length flag. The bytes included in the data CRC computation are the three index address marks, data address mark, and all data bytes.

The IBM 3740 sector transfer modules use a routine called "FNDFSEC" to position the head over the desired sector. The equivalent IBM System 34 module is called "FNDMSEC". The positioning is accomplished by continually reading ID fields until the correct sector is found. A return will be executed at approximately the third byte after the last ID CRC byte. This leaves plenty of time to set up to transfer the data field.

Sector skewing is unnecessary and undesirable when using the UDC. However, a provision is made to use a skew factor within "FNDFSEC" and "FNDMSEC". There is more than adequate time to perform all operations to read or write a sector, then find the following sector ID field before it passes. If a skew factor is not used, an entire track can be read in less than one revolution after the first sector is found.

If a skew factor is used, "FNDFSEC" or "FNDMSEC" should

be called with the number "1" stored in the memory location: "SKWFLG". Also, the address of the first entry of a skew conversion table must be stored in the memory location: "SKWTBL". The first value in this table is the physical sector number for logical sector 1, the second entry is the physical sector number for logical sector 2, etcetera.

6) Miscellaneous UDC Library Routines - The names and short definition of these routines are as follows.

a) "TXFER" - Transfers unlimited number of sectors.

b) "CMPCRCS" - Computes CRC values.

c) "CRCCHK" - Checks CRC values.

d) "PUSHRET" - Puts return address on stack.

Module (TXFER) - This module has all logic required to transfer an unlimited number of contiguous sectors to or from any system. This module must be called with the following memory locations set to the desired value.

a) ROUTADDR - Address of routine to read or write one sector

b) SECTOR - Sector number of first sector to be transferred

c) TRACK - Track number of first sector to be transferred

d) DTA - Address of first byte to be written or read

e) HDSOFT - Set to the number "1" if hard sectored disk is used, else set to the number "0"

f) SEC_TRK - Sectors per track

g) BYTSEC - Bytes per sector

h) NSECS - Number of sectors to be transferred

i) SKWFLG - Set to the number "1" if a skew factor is needed, else set to the number "0".

j) SKWTBL - If "SKWFLG" is "1" then this address must contain the address of the skew table.

"TXFER" can be used for any system; however the address of a module to read or write one sector must be given. This module can be a library module or otherwize. When this sector transfer module is called, the drive head will be over the correct track, but the correct sector must be found (correct sector number will always be in "SECTOR").

If the sector transfer module is a library "read" module then all CRC bytes for each sector must be precomputed and placed in contiguous memory starting at "CRCPTR". Library modules write two CRC bytes for each sector, therefore if a system requires only one CRC byte then every other byte of the CRC array must be a null or "don't care" value.

If the sector transfer module is a library "write" module then all disk CRC bytes will be placed in memory starting at "CRCPTR". Two CRC bytes will be written for each sector. If a system uses only one CRC byte then every other value in the CRC array will be a "don't care" value.

Module (CMPCRCS) - This UDC module computes a two byte CRC value for up to 128 sectors of data using the polynomial: X^16 + X^12 + X^5 + 1. Parameters must be set in the following memory locations.

a) DTA

b) NSECS

c) BYTSEC

Also, the "IX" register must be set to the initial 16 bit CRC value. If the IBM single density standard is used then this initial value should be BF84 hex.. If the IBM double density standard is used, then this value should be E295 hex.

All CRC bytes will be placed in memory starting at "CRCPTR". Memory is allocated for 256 CRC bytes.

Module (CRCCHK) - This module checks the CRC values for up to 128 sectors of data using the polynomial: X^16 + X^12 + X^5 + 1. Parameters are set similar to those for "CMPCRCS". The disk CRC values must be located in memory starting at "CRCPTR". The number of CRC errors will be placed in the memory location: "ERROR". No indication is given as to which sector had an error.

Module (PSHRET) - This module is used by the system routines to create a "CALREL" (call relative) instruction. The return address needed by "CALREL" is pushed onto the stack by "PSHRET". "CALREL" will be discussed in detail in the following section.

System Routines

The system routines are the middle level boxes of figure IV.1. The purposes of these routines are as follows.

a) Provide an interface between the file manipulation program and the UDC library.

b) Fill in any deficiencies of the UDC library.

The requirements of the system files are as follows.

a) The executable files must also be relocatable.

b) The third to eighth bytes must be the CPM file parameters.

c) The address of the stack pointer containing an abort address must be stored in "STKPTR". This is used by the UDC library routines for nested aborts when an error condition is discovered.

d) They must contain routines to read and write up to 128 contiguous sectors. These routines may be simply jump instructions to the appropriate UDC library modules.

Parameters passed to the system routines by "TRANSIT" are accomplished via the registers as shown below.

A - 0 if read, 1 if write

B - start track number

C - start logical sector number

D - Drive number (0 to 3)

E - number of sectors to transfer

HL - DTA (disk transfer address)

The first portion of all "SYSTEM.MAC" ("SYSTEM" can be any CP/M file name) files must follow the format shown on next page .

```
;FORMAT EXAMPLE FOR A SYSTEM.MAC FILE
;19 OCT 83

INCLUDE  LIBADDR.THF        ;THIS FILE CONTAINS ALL ADDRESSES FOR
                            ;THE GLOBAL UDCLIB ROUTINES IN THE
                            ;FORM OF EQUATES

     .Z80
     ASEG                   ;THE ASEG AND ORG STATEMENTS MUST BE
     ORG  100H              ;GIVEN EXACTLY AS SHOWN TO INSURE THE
                            ;FIRST BYTE OF THIS SYSTEM FILE IS
                            ;THE "JR   START" INSTRUCTION

;********************** CALL RELATIVE MACRO ******************
CALREL     MACRO     ADDRESS    ;"ADDRESS" IS THE ADDRESS OF
                                ;THE ROUTINE TO BE CALLED
           CALL      PSHRET     ;PUT THE CORRECT RETURN ADDRESS
                                ;ON THE STACK
           JR        ADDRESS
;************************************************************

SYSTEM:    JR        START      ;DON'T EXECUTE SYSTEM PARAMETERS

;SYSTEM CPM FILE PARAMETERS
REC_SEC:   DB        XX         ;RECORDS PER SECTOR
REC_TRK:   DB        XX         ;RECORDS PER TRACK
OFFSET:    DB        XX         ;DIRECTORY OFFSET
REC_DIR:   DB        XX         ;NUMBER OF DIRECTORY RECORDS
FILEK:     DW        XXXX       ;FILE STORAGE AREA (IN K)

START:     LD        (STKPTR),SP    ;SAVE FOR NESTED ABORTS
           PUSH      AF             ;SAVE READ/WRITE FLAG
           CALREL    INIT           ;INITIALIZE
           POP       AF             ;RESTORE READ/WRITE FLAG
           DEC       A              ;CHECK FOR READ OR WRITE
           JR        Z,WRITE        ;A WAS 1 IF WRITE IS NEEDED
           JR        READ

INIT:      ;ACCOMPLISH INITIALIZATION
           ;EXAMPLE: SAVE SECTOR NUMBER, TRACK NUMBER, DTA,
           ;AND NUMBER SECTORS IN "SECTOR", "TRACK", "DTA",
           ;AND "NUMSECS"

READ:      ;READ CONTIGUOUS SECTORS

WRITE:     ;WRITE CONTIGUOUS SECTORS

END
```

The definitions of the CPM file parameters are as follows.

a) REC_SEC - number of 128 byte blocks per disk sector

b) REC_TRK - number of 128 byte blocks per track

c) OFFSET - This is the number of records prior to the CPM directory. Use the following formula to compute "OFFSET".

OFFSET = [(DTRK) * (REC_TRK)] + DREC - 1

where: DTRK = track number of directory
DREC = record number of first directory record (normally 1)

d) FILEK - number of 1024 byte blocks available for file storage on disk (all space after directory)

The file manipulation program ("TRANSIT") will load the system routines into memory at one of two locations. Therefore, these routines were not and can not be designed to operate at a fixed memory location. This can be accomplished by making all jump instructions within the systems routine a "jump relative". If a "CALL" instruction is required from one location within the system routine to another location within the system routine, then the macro instruction "CALREL" must be used. This macro simply pushes the return address onto the stack by calling "PSHRET", then jumps relative.

Calls to the UDC library routines do not have to be relative calls because the library addresses are fixed. Addresses of the library routines are provided to the systems routines by the file named "LIBADDR.THF". This file should be included in all "SYSTEM.MAC" files.

The Microsoft Macro 80 Assembler should be used to assemble the "SYSTEM.MAC" files. Use the following command to link each system file.

```
L80 SYSTEM,SYSTEM.THF/N/E
```

The executable file created by this command must have the ".THF" extension or "TRANSIT" will be unable to load this file into memory.

File Manipulation Program

This program is written in the "C" programming language and uses the file name "TRANSIT". "TRANSIT" is capable of manipulating only CP/M files (refer to chapter II - CP/M File System). Figure IV.XX is a structure chart depicting the primary functions of "TRANSIT".

Module 1.1 (Initialize) - This module allocates memory then loads the system files and the UDC library. Memory is allocated as follows.

0100 - 3FFF	TRANSIT.COM
4000 - 4FFF	UDCLIB.THF
5000 - 57DF	Source SYSTEM.THF
57E0 - 57FF	RAM for SYSTEM.THF
5800 - 5FDF	Destination SYSTEM.THF
5FE0 - 5FFF	RAM for SYSTEM.THF
6000 - 9FFF	File Transfer Buffer

Module 1.2 (Acquire and Execute User's Request) - "TRANSIT" operates similar to the well known CP/M file transfer program "SWEEP". The user is prompted with the file name, then enters his request for one of the following functions.

a) Print destination directory

b) Delete source file

c) Tag a file (to be copied later)

d) Untag a file

e) Copy one file

f) Copy all tagged files

g) Copy all files

h) Load new "SYSTEM.THF" files

i) Print help menu

j) Exit to CP/M operating system

A description of the first five functions will be contained in this section. The last five functions are either simple one line "C" instructions, or loops that repeatedly call the "Copy" function. All source code is contained in appendix A.

Module (Print Destination Directory) - The destination directory is loaded into the file transfer buffer. The directory is then printed using the following algorithm. (Refer to figure II.8 for an example of a CP/M directory.)

1) Find next FCB with the first byte not equal to E5 (E5 signifies a deleted file) and 13th byte equal to 00 (00 signifies the first FCB of a file).

2) Print the file name and extension (bytes 2 to 12).

3) If not at end of directory, then go to step one.

Module (Delete Source File) - This module deletes all source FCB's corresponding to the prompting file name. The following algorithm is used.

1) Find next FCB with correct file name and first byte not equal to E5. If not found, then go to step 4.

2) Replace first byte of this FCB with E5.

3) If 16th byte is equal to 80H then go to step 1 else go to step 4. (If 16th byte is 80H then the FCB is full and it is possible for another FCB for this file to exist.)

4) Write memory's copy of the directory to disk.

Module (Tag) - Files are tagged by placing a one byte number ("tag") pointing to the first FCB into an array. This array is large enough to hold 50 tags. If the first FCB of a file is the first FCB of the directory, then the value of its tag is 0. If the first FCB of a file is the seventh FCB of the directory, then the value of its tag is 6, etcetera. A memory pointer to the first FCB of a tagged file can be computed using the following formula.

POINTER = DIRECTORY BASE + (32 * TAG)

Module (Untag) - A file is untagged by removing its tag from the tag array. This is accomplished by moving all tags after it down one position.

Module (Copy) - This is the work horse of "TRANSIT". It

transfers one file from the source disk to the destination disk. The following algorithm is used.

1) Scan all FCB's of the source directory to put all source group pointers corresponding to the file into a "Source Group Array". (Refer to chapter II - CP/M Files)

2) Load directory of destination system.

3) Build a destination disk allocation table (DAT). Each bit of the DAT corresponds to one group. If the group is used then the bit is set, else it is reset.

    a) Reset all bits of the DAT.

    b) Set DAT bits corresponding to groups used by the directory.

    c) Scan the destination directory to set all bits corresponding to file groups in use.

4) Scan the DAT to build an array of unused group pointers ("Destination Group Array") large enough to match the size of the "Source Group Array". If not enough unused group pointers available, then the file will not fit, so return with an error message.

5) Update memory's copy of the destination directory with FCB's representing the new file. If not enough room in directory then return with an error message.

6) Write memory's copy of the directory to the destination disk.

7) Read up to 16 groups of the source file into the file transfer buffer using track and sector numbers corresponding to the "Source Group Array".

8) Write the contents of the file transfer buffer to the destination disk using tracks and sectors corresponding to the "Destination Group Array".

9) If the number of groups remaining to be transferred is greater than zero, then go to step 7.

10) Reload the source directory.

The process of converting group pointers to actual

track and sector numbers is accomplished within a module called "Disk IO". This module has the responsibility of providing an interface to the "SYSTEM.THF" routines. The parameters needed to do the conversion are acquired from the "SYSTEM.THF" routines. These parameters are "offset", "records per track", and "records per sector".

The formulas to acquire track and sector numbers are as follows.

```
noffrec = (grpnum * 8) + offset

track = noffrec / rec_trk

sector = [(noffrec mod rec_trk) + rec_sec] / rec_sec

(use integer arithmetic)
```

where:

noffrec = Record number of first record in group if there was no offset. (Offset is the number of records on disk prior to the directory.)

grpnum = group number

rec_trk = records per track

rec_sec = records per sector

V. Recommendations and Conclusions

The development of the Universal Disk Controller was a successful undertaking. All but one of the original requirements were met. All other limitations of the overall disk transfer system are due to the bare drive limitations and software limitations.

Six ways to improve the current file transfer system are given below.

a) Add 8 inch double density.

b) Add an 80 track double sided 5 inch drive.

c) Increase the number of microcomputer systems incorporated by enhancing software.

d) Enhance software to include multiple operating systems.

e) Change the Basic Input/Output System Software (BIOS) to use the UDC.

f) Put the UDC library in ROM.

The only significant limitation of the UDC is its inability to transfer 8 inch double density formats. This can be remedied in future developments by interfacing the UDC to a Z80 based microcomputer with at least a 6 MHZ CPU clock. The maximum data rate of the Z80 SIO with its current 4 MHz CPU is 700 K bits per second. 1000 K bits per second is required to transfer 8 " double density.

The current 5 inch drives in use are 40 track single sided drives. By adding an 80 track double sided drive, many

more microcomputer systems could be added to the system. No changes to the UDC would be required; however the UDC library would have to be updated to check the side flag in the track ID field of soft sectored disks.

The systems currently implemented are IBM single density, Northstar Horizon, and NEC 8000. This limited number of systems leaves a lot of room for additions.

The file transfer system is currently capable of transferring only CP/M files. This limition is due only to "TRANSIT.C". This program could be enhanced to accommodate multiple operating systems.

There are presently four 8" drives used in the system. Two are for use with the UDC and two are used by the home system's disk controller and operating system. The second two drives would not be required if the BIOS were changed to use the UDC and its drives. An alternative to this suggestion would be to add circuitry to the UDC to multiplex the 8" drives between the two disk controllers.

A ROM chip could be added to the UDC to house the UDC library. This could save much random access memory in the home system. However, it would not be feasible to accomplish this enhancement until the UDC was in use long enough to thoroughly test its software.

The next disk controller designed at AFIT should involve a long term study to create an IDC (Intelligent Disk Controller). This controller should have its own high speed CPU, ROM, and RAM. The CPU should be fast enough to

accommodate 8" double density formats. The ROM would house the current library routines, most popular systems routines, and routines to interface with the outside world. The RAM would be used as sector registers, track registers, and general purpose use by the firmware. This IDC should be S-100 bus compatible.

The current Universal Disk Controller and its associated software show the Intelligent Disk Controller is feasible and can be designed and built at AFIT now.

Bibliography

Gibson, Glenn A. and Yucheng Liu. Microcomputers For Engineers and Scientists. Engle Cliffs: Prentice Hall, 1980.

Harman, Jefferson H. "IBM Compatible Disk Drives," Byte Magazine, 4 (10): 100-106 (October 1979).

Head, Gene. "CP/M Exchange," Dr. Dobbs Journal, 77: 50-52 (March 1983).

Hoeppner, John F. and Larry H. Wall. "Encoding/Decoding Techniques Double Floppy Disks Capacity," Computer Design, 19 (2): 127-135 (February 1980).

Hogan, Thom. Osborn CP/M. Berkley: McGraw Hill, 1982.

Lancaster, Don. TTL Cookbook. Indianapolis: Howard W. Sams & Co., 1974.

Nicholson, James and Roger Camp. "Build a Super Simple Floppy Disk Interface Part I," Byte Magazine, 6 (5): 360-374 (May 1981).

Shugart(A). 5.25 Inch Format Manual. Sunnyvale: Shugart, 1981.

Shugart(B). 8 Inch Format Manual. Sunnyvale: Shugart, 1981.

Shugart(C). SA400 OEM Manual. Sunnyvale: Shugart, 1981.

Shugart(D). SA400 Service Manual. Sunnyvale: Shugart, 1981.

Shugart(E). SA800/801 OEM Manual. Sunnyvale: Shugart, 1981.

Shugart(F). SA800/801 Service Manual. Sunnyvale: Shugart, 1981.

Shugart(G). Application Note for the Design of a Floppy Disk Controller. Sunnyvale: Shugart, 1981.

Western Digital. 1983 Components Handbook.

Zaks, Rodney and Austin Lesea. Microprocessor Interfacing Techniques. Sybex Inc. 1979.

Zilog. Zilog Data Book. Cambell: Zilog, 1982.

```
/****************************************************************************/
/* PROGRAM NAME: TRANSIT.C                                                  */
/* DATE: 28 OCT  83                                                         */
/* PURPOSE: READ, WRITE, AND DELETE CPM 80 FILES USING THE UDC              */
/****************************************************************************/
/*                                                                          */
/* definitions:                                                             */
/*                                                                          */
/*      A record is a 128 byte block.                                       */
/*      A sector is the smallest block transferrable to a disk.             */
/*      A group is 8 records (1k).                                          */
/*      A group pointer is one entry in the disk file control block.        */
/*                                                                          */
/* NOTE:                                                                    */
/*      This program allows for a max file size of 128k.                    */
/*      This restriction is due only to the size of the source and          */
/*      destination group arrays.                                           */
/*                                                                          */
/****************************************************************************/

/********* MEMORY MAP ***********/
/*                              */
/* 0100 - 3FFF    TRANSIT.COM   */
/* 4000 - 4FFF    UDCLIB.THF    */
/* 5000 - 57DF    SOURCE.THF    */
/* 5800 - 5FFF    DESTIN.THF    */
/* 6000 - 9FFF    TRANSFER BUFF */
/*                              */
/********************************/


#include "defines.h"            /* file with constant definitions */

/************************** GLOBAL VARIABLES: ****************************/

char *DTA;                      /* disk transfer address */
char system[numsystems][15];    /* name of com files for each system */
int sysaddr[2];                 /* addresses of com files */
char drive[2];                  /* source & destination drive */
char group[2][maxfilesize];     /* group array for destination and source */
```

```
char error;                     /* general purpose error flag */
char dirnum;                    /* directory number */
char tagfile[51];               /* directory numbers of tagged files */
char numtags;                   /* number of tagged files */
char DAT[maxdisk / 8];          /* disk allocation table */
                                /* one bit (flag) for each k of dsk space */
int destK;                      /* space used by files on destination disk */
int fileK;                      /* file size of most resent tagged file */
int taggedK;                    /* total size (in K) of all tagged files */
int FCBptr[maxFCBs];            /* pointers to all source FCBs */
int numFCBs;                    /* number of file control blocks in file */
int numfilgrps;                 /* number of groups in the file */
int numfilerec;                 /* number records in file */
int DATsize;                    /* number bytes in disk alloc table */

/************* parameters acquired from system .com files ********************/

char rec_sec[2];                        /* records per sector */
char rec_trk[2];                        /* sectors per track */
char offset[2];                         /* # records prior to directory */
char rec_dir[2];                        /* # records in full directory */
int  d_disk_k;                          /* number of k available for file */
                                        /* storage on destination disk    */
```

```
/*************************************************************/
/*      DATE:  18 NOV 83                                     */
/*      FUNCTION NAME:  main                                 */
/*      PURPOSE:  CALL ALL SUPPORTING FUNCTONS               */
/*      GLOBALS USED: dirnum                                 */
/*      GLOBALS CHANGED: none                                */
/*      MODULES CALLED: just about all                       */
/*************************************************************/

main()
{
char input;
char backflag;          /* flags user input to backup one file */

init();
backflag = 0;

while(0==0)             /* do until ^C is entered */
        {
        if(backflag == 1)       /* back up one file */
                {
                backup();
                backflag = 0;
                }
        else
                nextfile();     /* print next file name and get its dirnum */

        if(error == 1)          /* no files */
                {
                init();         /* start over */
                nextfile();
                }
        input = toupper(getchar());

        switch(input)
                {
                case 'A':
                        txferall();
                        init();         /* start over */
                        break;

                case 'B':               /* backup */
                        backflag = 1;
                        break;

                case 'C':               /* copy */
                        copy(dirnum);
                        break;

                case 'D':               /* print destination directory */
                        getdir(destin);
                        printdirec();
                        getdir(source);
                        break;
```

```
					case 'K':
						kill();		/* delete source directory entry */
						break;

					case 'M':			/* copy all tagged files */
						masscopy();
						break;

					case 'R':			/* reset systems */
						init();
						break;

					case 'T':
						tag();
						break;

					case 'U':
						untag();
						break;

					case 'X':
						exit();

					case '?':
						menu();
						break;

					default:
						puts("\b \b");		/* erase input char */

					}
				}
			}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                                */
/*      FUNCTION NAME:  init                            */
/*      PURPOSE: initialization                         */
/*      INPUTS/OUTPUTS: none                            */
/*      GLOBALS CHANGED: numtags, taggedK, dirnum       */
/*      MODULES CALLED:  readfile, UDCLIB (asm routine) */
/*      CALLING MODULES: main                           */
/*******************************************************/

init()
{
strcpy(system[0],"NEC.THF");
strcpy(system[1],"IBM.THF");
strcpy(system[2],"H895S.THF");
strcpy(system[3],"KAYPRO.THF");
strcpy(system[4],"NS.THF");

sysaddr[source] = srcaddr;      /* assign assembly routine addresses */
sysaddr[destin] = destaddr;

readfile("UDCLIB.THF",libaddr);

calla(pioinit,0,0,0,0);         /* initialize PIO and turn on drive motors */

getsystems();                   /* load system routines into memory */

numtags = 0;                    /* nothing tagged yet */
taggedK = 0;

dirnum = -1;                    /* first directory number */

}
```

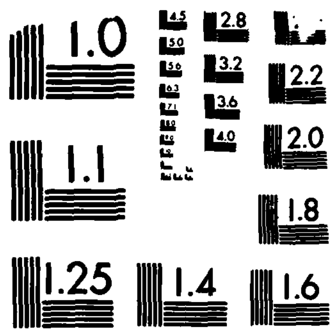
```
/***********************************************************/
/*      DATE:  18 NOV 83                                  */
/*      FUNCTION NAME:  getsystems                        */
/*      PURPOSE: display systems menu, acquire desired    */
/*               systems and drive from user              */
/*      INPUTS/OUTPUTS: none                              */
/*      GLOBALS CHANGED: drive[], system, rec_sec,        */
/*                       rec_trk, offset, rec_dir,        */
/*                       d_disk_k                         */
/*      MODULES CALLED:  getdrive, readfile, getdir,      */
/*                       compdestK, menu                  */
/*      CALLING MODULES: init                             */
/***********************************************************/


getsystems()
{
char userinp;
char srcnum;                    /* source disk system */
char desnum;                    /* destination disk system */

clearcrt();

puts("\n************************  SYSTEMS  ***********************\n\n");
puts("     0) NEC\n");
puts("     1) IBM 8\n");
puts("     2) H89 5 soft\n");
puts("     3) KAYPRO\n");
puts("     4) NORTHSTAR\n\n");
puts("**************************************************************\n\n");


printf("\nENTER SOURCE SYSTEM (0 - %d)? ",numsystems-1);

do
        userinp = getchar();
while
        (userinp < '0' || userinp >= '0' + numsystems);

srcnum = userinp-48;            /* convert ascii to number */

readfile(system[srcnum], srcaddr);

printf("\n\nSOURCE DRIVE");

drive[source] = getdrive();
```

```
printf("\nENTER DESTINATION SYSTEM (0 - %d)? ",numsystems-1);

do
        userinp = getchar();
while
        (userinp < '0' || userinp >= '0' + numsystems);

desnum = userinp-48;            /* convert ascii to number */

readfile(system[desnum], destaddr);

printf("\n\nDESTINATION DRIVE");
drive[destin] = getdrive();

puts("You may now change disks as necessary\nPress any key when ready ");
getchar();

/* load parameters from system .THF files */
rec_sec[source] = peek(srcaddr + 2);
rec_trk[source] = peek(srcaddr + 3);
offset[source] = peek(srcaddr + 4);
rec_dir[source] = peek(srcaddr + 5);

rec_sec[destin] = peek(destaddr + 2);
rec_trk[destin] = peek(destaddr + 3);
offset[destin] = peek(destaddr + 4);
rec_dir[destin] = peek(destaddr + 5);
d_disk_k = peek(destaddr + 6) + (256 * peek(destaddr + 7));

getdir(destin);         /* initialize number of K in destination disk */
compdestK();

menu();

getdir(source);
}
```

AD-A138 333 UNIVERSAL DISK CONTROLLER FOR MICROCOMPUTERS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AF OH SCHOOL OF ENGINEERING F N ELAM DEC 83 AFIT/GE/EE/83D-20 2/2

UNCLASSIFIED F/G 9/2 NL

END
FILMED
4-84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
/*****************************************************/
/*      DATE:  18 NOV 83                              */
/*      FUNCTION NAME: menu                           */
/*      PURPOSE:  print menu on CRT                   */
/*      CALLING MODULES: main, init                   */
/*****************************************************/


menu()
{
char i;

clearcrt();

for(i = 0; i < 49; ++i)
        putchar('-');

puts("\n!\tA   TRANSFER ALL\tR   RESTART\t!\n");
puts("!\tB   BACKUP\t\tT   TAG\t\t!\n");
puts("!\tC   COPY\t\tU   UNTAG\t!\n");
puts("!\tD   DEST. DIRECTORY\tX   EXIT\t!\n");
puts("!\tM   MASS COPY\t\t?   MENU\t!\n");
puts("!\tK   KILL/DELETE SOURCE FILE\t\t!\n");

for(i = 0; i < 49; ++i)
        putchar('-');

puts("\n\n");
}
```

```
/****************************************************/
/*      DATE:  18 NOV 83                            */
/*      FUNCTION NAME: nextfile                     */
/*      PURPOSE: change parameters as neccesary to view */
/*               filename on source disk            */
/*      INPUTS: none, uses globals                  */
/*      GLOBALS USED: dirnum, fileK, numtags, tagfile[] */
/*      GLOBALS CHANGED: dirnum, fileK              */
/*      MODULES CALLED: pfilename, computeK         */
/*      CALLING MODULES: main                       */
/****************************************************/


nextfile()
{
char i,j,character;
char startpt;   /* used to track when the entire directory has been checked */
char *dirptr;   /* directory pointer */


startpt = dirnum;

putchar('\n');

while(0==0)                /* stay in loop till a break condition is found */
        {
        ++dirnum;

        dirptr = direcbase + 32*dirnum;         /* ptr to first FCB char */

        if(peek(dirptr) != 0xE5 && peek(dirptr+12)==0)
                {
                error = 0;              /* found next entry */
                break;
                }
        else if(peek(dirptr+1) == 0xE5)         /* at end of directory */
                {
                dirnum = -1;    /* start at the top of the directory */
                putchar('\n');  /* double space between filenames */
                }

        if(dirnum == startpt)   /* looked at entire direc w/o finding file */
                {
                error = 1;
                break;
                }
        }
```

```
if(error == 1)
        return;

pfilename(dirnum);              /* print the filename pointed to by dirptr */

for(i = 1; i < 13; ++i)         /* step to ruler line */
        if(peek(dirptr + i) == ' ')
                putchar(' ');

computeK(dirptr);               /* compute filesize */

printf(" %3dK:",fileK);

for(i = 0; i < numtags; ++i)            /* put a 'T' if tagged */
        if(tagfile[i] == dirnum)
                {
                putchar('T');
                break;
                }
}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME: backup                          */
/*      PURPOSE: opposit function of 'nextfile'        */
/*      PURPOSE: change parameters as neccesary to view */
/*              previos filename on source disk        */
/*      INPUTS: none, uses globals                     */
/*      GLOBALS USED: dirnum, fileK, numtags, tagfile[] */
/*      GLOBALS CHANGED: dirnum, fileK                 */
/*      MODULES CALLED: pfilename, computeK            */
/*      CALLING MODULES: main                          */
/*******************************************************/


backup()
{
char i,j,character;
char *dirptr;           /* directory pointer */

putchar('\n');

while(0==0)             /* stay in loop till a break condition is found */
        {
        if(dirnum == 0)
                {
                dirnum = rec_dir[source] * 4;   /* move to end of directory */
                putchar('\n');                  /* double space */
                }
        --dirnum;

        dirptr = direcbase + 32*dirnum;         /* ptr to first FCB char */

        if(peek(dirptr) != 0xE5 && peek(dirptr+12)==0)
                break;                          /* found a FCB in use */
        }

pfilename(dirnum);              /* print the filename pointed to by dirptr */

for(i = 1; i < 13; ++i)         /* step to ruler line */
        if(peek(dirptr + i) == ' ')
                putchar(' ');

computeK(dirptr);               /* compute filesize */
printf(" %3dK:",fileK);

for(i = 0; i < numtags; ++i)
        if(tagfile[i] == dirnum)
                {
                putchar('T');
                break;
                }
}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME:  pfilename                      */
/*      PURPOSE: print a file name                     */
/*      INPUTS: dirnum                                 */
/*      OUTPUTS: none                                  */
/*      CALLING MODULES: nextfile, backup, txferall,   */
/*                       masscopy                      */
/*******************************************************/


pfilename(dirnum)
char dirnum;
{
char i;
char *dirptr;

dirptr = direcbase + 32*dirnum + 1;

for(i = 1; i < 9; ++i)          /* print the primary filename */
        {
        if(*dirptr != ' ')
                putchar(*dirptr);
        ++dirptr;
        }

putchar('.');                   /* period prior to extension name */

for(i = 0; i < 3; ++i)          /* print extension name */
        {
        if(*dirptr != ' ')
                putchar(*dirptr);
        ++dirptr;
        }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                              */
/*      FUNCTION NAME: tag                            */
/*      PURPOSE: tag a file                           */
/*      INPUTS/OUTPUTS: none                          */
/*      GLOBALS USED:  numtags, tagfile[], dirnum,    */
/*                     taggedK, fileK                 */
/*      GLOBALS CHANGED: numtags, tagfile[], taggedK  */
/*      CALLING MODULES:  main                        */
/*****************************************************/


tag()
{
char i;

if(numtags == 50)
        {
        puts("\b Sorry, only 50 tagged files allowed");
        return;
        }

for(i = 0; i < numtags; ++i)
        if(tagfile[i] == dirnum)
                {
                puts("\b ");            /* erase the 'T' */
                return;                 /* already tagged */
                }

tagfile[numtags] = dirnum;

++numtags;

taggedK = taggedK + fileK;

printf(" (%dK tagged      %dK remaining)",taggedK,d_disk_k-taggedK-destK);
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME: untag                         */
/*      PURPOSE: untag a file                        */
/*      INPUTS/OUTPUTS: none                         */
/*      GLOBALS USED:  numtags, tagfile[], dirnum,   */
/*                     taggedK, fileK                */
/*      GLOBALS CHANGED: numtags, tagfile[], taggedK */
/*      CALLING MODULES:  main                       */
/*****************************************************/


untag()
{
char i;

for(i = 0; i < numtags; ++i)
        if(tagfile[i] == dirnum)
                {
                --numtags;

                while(i < numtags)      /* shift array down */
                        {
                        tagfile[i] = tagfile[i+1];
                        ++i;
                        }

                puts("\b\b  ");         /* rubout the 'T' */
                taggedK = taggedK - fileK;
                printf(" (%dK tagged      %dK remaining)",
                        taggedK,d_disk_k-taggedK-destK);
                break;
                }
}
```

```
/**************************************************/
/*      DATE:  18 NOV 83                          */
/*      FUNCTION NAME: txferall                   */
/*      PURPOSE: copy all files from source to destin  */
/*      INPUTS:  none, uses globals               */
/*      OUTPUTS: none                             */
/*      GLOBALS USED:  dirnum                     */
/*      GLOBALS CHANGED:  dirnum                  */
/*      MODULES CALLED: pfilename, copy           */
/*      CALLING MODULES: main                     */
/**************************************************/


txferall()
{
char *dirptr;

dirptr = direcbase;

for(dirnum = 0; peek(dirptr+1) != 0xE5; ++dirnum)
        {
        dirptr = direcbase + 32*dirnum;         /* ptr to first FCB char */

        if(peek(dirptr) != 0xE5 && peek(dirptr+12)==0)
                {
                puts("\n\t\tCOPYING ");
                pfilename(dirnum);
                copy(dirnum);                   /* found next entry */
                }
        }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  masscopy                     */
/*      PURPOSE: copy all tagged files               */
/*      INPUTS/OUTPUTS: none                         */
/*      GLOBALS USED:  dirnum                        */
/*      GLOBALS CHANGED:  dirnum                     */
/*      MODULES CALLED: pfilename, copy              */
/*      CALLING MODULES:  main                       */
/*****************************************************/


masscopy()
{
char i;

if(numtags == 0)
        {
        puts("\bYou need to tag a file");
        return;
        }

putchar('\n');

for(i = 0; i < numtags; ++i)
        {
        puts("\n\t\tCOPYING ");
        pfilename(tagfile[i]);
        copy(tagfile[i]);
        }

putchar('\n');

taggedK = 0;            /* nothing tagged now */
numtags = 0;
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  copy                         */
/*      PURPOSE: transfer one file from source to dest */
/*      INPUTS:  dirnum                              */
/*      OUTPUTS: none                                */
/*      GLOBALS USED:  dirnum,numfilegrps            */
/*      GLOBALS CHANGED:  numfilegrps                */
/*      MODULES CALLED: finddirentry, getdir, delete, */
/*                      makeDAT, setgrps, updir,     */
/*                      writedir, txfer              */
/*      CALLING MODULES: main, txferall, masscopy    */
/*****************************************************/


copy(dirnmb)
char dirnmb;
{
char direntry[13];              /* directory entry */
char i;

for(i = 0; i < 13; ++i)         /* assign character to direntry */
        direntry[i] = peek(direcbase + dirnmb * 32 + i);

finddirentry(direntry);    /* find directory entry in source directory then */
                           /* put all source group pointers into an array */

getdir(destin);            /* load directory of destination disk to direcbase */

if(delete(direntry) == 1)       /* delete file if exists on destination disk */
        {                       /* returns 1 if user don't want to delete it */
        getdir(source);         /* keep source directory in memory */
        return;                 /* user does not want to transfer this file */
        }

if(numfilgrps != 0)             /* if not a 0 K file */
        {
        makeDAT();      /* build disk allocation table for destination disk */
        setgrps();      /* put group numbers for destin disk into an array */
        if(error)
                {
                printf("%c Sorry, will not fit\n\n",beep);
                getdir(source);
                return;
                }
        }
```

```
updir(direntry);   /* put new entry in destination directory (in RAM only) */

if(error)
        {
         puts(" Sorry, not enough room in directory\n\n");
         getdir(source);
         return;
        }

writedir(destin);        /* write directory with new entry to disk */

if(numfilgrps != 0)      /* if not 0 K file */
        txfer();         /* transfer the file */

getdir(source);          /* keep source directory in memory */

}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME:  txfer                          */
/*      PURPOSE:  nitty gritty routine used by 'copy'  */
/*      GLOBALS USED:  DTA                             */
/*      GLOBALS CHANGED: DTA                           */
/*      MODULES CALLED:  grptxfer                      */
/*      CALLING MODULES:  copy                         */
/*******************************************************/


txfer()
{
int grpnum;

grpnum = 0;             /* transfer all groups starting with the first one */

while(grpnum < numfilgrps)
        {
        DTA = filebuff;                 /* fill up the buffer with source */
        grptxfer(source,read,bufsize,grpnum);
        DTA = filebuff;                 /* write the buffer to destination */
        grptxfer(destin,write,bufsize,grpnum);
        grpnum = grpnum + bufsize;      /* prep for next buffer fillup */
        }

verify();
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  grptxfer                     */
/*      PURPOSE:  read or write a finite number of   */
/*                groups from source or to destination */
/*      INPUTS:  src_dest, rd_wrt, buffersz, grpnum  */
/*      MODULES CALLED:  disk_IO                     */
/*      CALLING MODULES:  txfer, verify              */
/*****************************************************/


grptxfer(src_des,rd_wrt,buffersz,grpnum)
char src_des,rd_wrt,buffersz;
int grpnum;
{
char numbufgrps;
char contig;

numbufgrps = 0;         /* used to track number of groups in buffer */

while(numbufgrps < buffersz && grpnum < numfilgrps)
      {
      contig = 1;                /* number of contiguous groups */

      while(group[src_des][grpnum+contig-1] == group[src_des][grpnum+contig]-1
               && numbufgrps + contig < buffersz)

               ++contig;            /* compute number of contiguous groups */

      disk_IO(src_des,rd_wrt,group[src_des][grpnum],contig*8);

      grpnum = grpnum + contig;       /* prep for next disk txfer */

      numbufgrps = numbufgrps + contig;
      }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  verify                       */
/*      PURPOSE:  verify file was copied correctly   */
/*      GLOBALS USED:  numfilgrps, DTA               */
/*      GLOBALS CHANGED: DTA                         */
/*      MODULES CALLED:  grptxfer                    */
/*      CALLING MODULES: txfer                       */
/*****************************************************/


verify()
{
int grpnum;
int numcompares;          /* number of bytes in buffer */
char match;
char *buff2;              /* second buffer for destin file */

buff2 = filebuff + (bufsize/2) * 1024;  /* split filebuff into two parts */

grpnum = 0;               /* transfer all groups starting with the first one */

puts("\n\t\tVERIFYING");

while(grpnum < numfilgrps)
        {
        DTA = filebuff;                 /* fill up 1st buffer with source */
        grptxfer(source,read,bufsize/2,grpnum);
        numcompares = DTA - filebuff;
        DTA = buff2;                    /* fill 2nd buffer with destination */
        grptxfer(destin,read,bufsize/2,grpnum);

        /* use "compare" in UDCLIB to compare the buffers */
        match = calla(compare,0,filebuff,numcompares,buff2);

        if(match)
                grpnum = grpnum + bufsize/2;
        else            /* recopy the one buffer which has the error */
                {
                puts("\n\t\tTRANFER ERROR, RECOPYING");
                DTA = filebuff;
                grptxfer(destin,write,bufsize/2,grpnum);
                }
        }

}
```

```
/****************************************************/
/*      DATE: 18 NOV 83                             */
/*      FUNCTION NAME:  kill                        */
/*      PURPOSE:  delete one file                   */
/*      GLOBALS USED:  dirnum                       */
/*      MODULES CALLED:  compare(asm routine), writedir */
/*                       dselect(assembly routine)  */
/*      CALLING MODULES:  main                      */
/****************************************************/


kill()
{
char *dirptr;
char *frstFCB;
char match;

puts("ill/delete (Y or N)? ");
if(getchar() == 'Y')
        {
        frstFCB = direcbase + 32 * dirnum;
        dirptr = frstFCB + 32;

        while(peek(dirptr + 1) != 0xE5)
                {
                match = calla(compare,0,frstFCB,12,dirptr);
                if(match)
                        *dirptr = 0xE5;         /* delete one FCB */
                dirptr = dirptr + 32;           /* point to next FCB */
                }
        *frstFCB = 0xE5;        /* delete first FCB */
        writedir(source);       /* put directory with deleted FCB's on disk */
        }

calla(dselect,0,0,0,0);         /* deselect all drives */
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  delete                       */
/*      PURPOSE:  delete a destination file to permit */
/*                rewriting it                       */
/*      INPUTS:  direntry                            */
/*      OUTPUTS: 0 if deleted, 1 if user doesn't want */
/*               to delete the file                  */
/*      MODULES CALLED:  compare (assembly routine)  */
/*      CALLING MODULES:  copy                       */
/*****************************************************/


delete(direntry)
char direntry[];
{
char *dirptr;
char firstFCB;          /* flag */
char response;
char match;             /* 0 if string don't match, 1 if they do */

firstFCB = 1;

for(dirptr = direcbase; peek(dirptr+1) != 0xE5; dirptr = dirptr + 32)
        {
        match = calla(compare,0,direntry,12,dirptr);
        if(match)
                {
                if(firstFCB)
                        {
                        firstFCB = 0;

                        puts(" ***file exists, delete (Y or N)? ");

                        do
                                {
                                response = toupper(getchar());
                                putchar('\b');
                                }
                        while(response != 'Y' && response != 'N');

                        if(response == 'N')
                                return(1);  /* 1 flags request not to delete */
                        }
                *dirptr = 0xE5;         /* delete this FCB */
                }
        }
return(0);
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                              */
/*      FUNCTION NAME:  getdir                        */
/*      PURPOSE:  load directory from disk            */
/*      INPUTS: src_des                               */
/*      GLOBALS USED:  DTA                            */
/*      GLOBALS CHANGED: DTA                          */
/*      MODULES CALLED:  disk_IO, compare (asm routine) */
/*      CALLING MODULES:  main, getsystems, copy      */
/*****************************************************/


getdir(src_des)
char src_des;           /* system (either source or destination) */
{
char *dir2;             /* pointer to 2nd copy of directory */
char result;

do                      /* read 2 copies of directory until they match */
        {
        DTA = direcbase;
        disk_IO(src_des,read,0,rec_dir[src_des]); /* 0 is grp # of directory */
        dir2 = DTA;

        disk_IO(src_des,read,0,rec_dir[src_des]);   /* 2nd copy of directory */

        /* verify good directory read */
        result = calla(compare,0,direcbase,DTA-dir2,dir2);

        if(result == 0)
                puts("\n***fixing directory read error");
        }
while(result == 0);

poke(dir2,0xE5);
poke(dir2+1,0xE5);              /* flags end of directory */
calla(dselect,0,0,0,0);         /* deselect all drives */
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  writedir                     */
/*      PURPOSE: write updated directory to dest disk  */
/*      INPUTS:  src_des                             */
/*      GLOBALS USED:  DTA                           */
/*      GLOBALS CHANGED:  DTA                        */
/*      MODULES CALLED:  disk_IO                     */
/*      CALLING MODULES:  kill, copy                 */
/*                                                   */
/*****************************************************/


writedir(src_des)       /* WRITE UPDATED DIRECTORY TO DESTINATION DISK */
char src_des;
{
char *dir2;             /* pointer to 2nd copy of directory */
char result;

compdestK();          /* update number of K used by files on destination disk */

do                     /* write then read back directory to verify */
        {
        DTA = direcbase;
        disk_IO(src_des,write,0,rec_dir[src_des]);
        dir2 = DTA;
        disk_IO(src_des,read,0,rec_dir[src_des]);   /* read what was written */

        /* verify good direcectory write */
        result = calla(compare,0,direcbase,DTA-dir2,dir2);

        if(result == 0)
                puts("\n***fixing directory write error");
        }
while(result == 0);             /* bad write if result == 0 */

poke(dir2,0xE5);
poke(dir2+1, 0xE5);             /* mark end of directory */
}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME:  updir                          */
/*      PURPOSE:  update destination directory         */
/*      INPUTS:  direntry                              */
/*      GLOBALS USED:  DTA                             */
/*      GLOBALS CHANGED:  DTA                          */
/*      MODULES CALLED:  compare(asm routine)          */
/*      CALLING MODULES:  copy                         */
/*******************************************************/


updir(direntry)                 /* UPDATE DESTINATION DIRECTORY */
char direntry[];
{
char *dir_ptr;
int grpnum;
int i,j,k;

i = direcbase;                  /* initialize directory ptr */
j = 0;                          /* initialize direc blk counter */
grpnum = 0;                     /* initialize array num in grp array */

while((i < direcbase + rec_dir[destin] * 128) && (j < numFCBs))
      {
       if(peek(i) == 0xE5)
          {
           dir_ptr = i;
           for(k = 0; k < 12; ++k)
               {
                *dir_ptr = direntry[k];  /* fill with 0 + source name */
                ++dir_ptr;
               }
           *dir_ptr = j;   /* 13th byte in directory is the FCB number */
           ++dir_ptr;                    /* point to next byte */
           *dir_ptr = 0;                 /* next two bytes are zero */
           ++dir_ptr;
           *dir_ptr = 0;
           ++dir_ptr;                    /* point to # of records byte */
           if(j < numFCBs -1)
                *dir_ptr = 128;          /* 128 records in full dir grp */
           else
                *dir_ptr = numfilerec % 128;     /* last gets left over */
           ++dir_ptr;                            /* point to grp field */
```

```
                    for(k = 0; k < 16; ++k)
                        {
                         if(grpnum < numfilgrps)
                              {
                               *dir_ptr = group[destin][grpnum];
                               ++grpnum;
                              }
                         else
                              *dir_ptr = 0;
                         ++dir_ptr;
                        }
                    ++j;
                   }
              i = i + 32;                    /* look at next dir blk */
             }

    if(j < numFCBs)
            error = 1;                       /* global error flag */
    else
            error = 0;

    }
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME: finddirentry                    */
/*      PURPOSE: find directory entry in source direct */
/*      INPUTS:  direntry                              */
/*      GLOBALS USED:  FCBptr, numFCBs, numfilerec,    */
/*                     numfilgrps, group[]             */
/*      GLOBALS CHANGED:  all that's used              */
/*      MODULES CALLED:  compare(asm routine)          */
/*      CALLING MODULES:  copy                         */
/*******************************************************/


finddirentry(direntry)
char direntry[];
{
char *dirptr;                       /* directory pointer */
char match;                         /* 0 if strings don't match, 1 if they do */
int i,j;

/* Search the directory for a match. Each time find match store ptr to it */

j = 0;                     /* initialize # of file control blocks in file */

for(dirptr = direcbase; peek(dirptr+1) != 0xE5; dirptr = dirptr+32)
         {
          match = calla(compare,0,direntry,12,dirptr);
          if(match)
               {
                FCBptr[j] = dirptr;
                ++j;
                if(peek(dirptr + 15) != 0x80)
                        break;              /* last FCB if not 80H records */
               }
         }
numFCBs = j;

/* compute number of records in file */
numfilerec = (numFCBs - 1)* 128 + (peek(FCBptr[numFCBs - 1] + 15));

numfilgrps = (numfilerec + 7) / 8;       /* compute number groups in file */

/* get source file block #'s */
j = 0;                              /* initialize source group array index */
for(i = 0; i < numFCBs; ++i)
        for(dirptr = FCBptr[i] + 16; dirptr < FCBptr[i] + 32; ++dirptr)
                {
                group[source][j] = *dirptr;
                ++j;
                }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME: printdirec                    */
/*      PURPOSE: put destination directory on CRT    */
/*      INPUTS:  direntry                            */
/*      GLOBALS USED:  destK, d_disk_k, destK        */
/*      GLOBALS CHANGED:  none                       */
/*      MODULES CALLED:  clearcrt                    */
/*      CALLING MODULES:  main                       */
/*****************************************************/


printdirec()
{
int i, j, k;

clearcrt();

for(i = 0; i < 62; ++i)
        putchar('-');

putchar('\n');

    k = 0;                              /* k tracks when to give a CR LF */

    for(i = direcbase; peek(i+1) != 0xE5; i = i+32)
        if(peek(i) != 0xE5 && peek(i + 12) == 0)
           {
            for(j = 1; j < 12; ++j)
                putchar(peek(i + j));           /* print one file name */
            printf("  :  ");                    /* seperate file names */
            ++k;
            if(k % 4 == 0)                      /* ret every 4th name */
                putchar('\n');
           }

printf("\n\n     *** %dK used    ***%dK remaining\n",destK,d_disk_k-destK);

for(i = 0; i < 62; ++i)
        putchar('-');

putchar('\n');
}
```

```
/*********************************************************/
/*      DATE:  18 NOV 83                                 */
/*      FUNCTION NAME:  compdestK                        */
/*      PURPOSE: compute file space used on destin disk */
/*      GLOBALS USED:  destK                             */
/*      GLOBALS CHANGED:  destK                          */
/*      CALLING MODULES:  writedir, getsystems           */
/*********************************************************/

/* destination directory must be loaded prior to calling this routine */

compdestK()
{
char *dirptr;

dirptr = direcbase;
destK = 0;

while(peek(dirptr + 1) != 0xE5)
        {
        if(peek(dirptr) != 0xE5)
                destK = destK + (peek(dirptr+15) + 7) / 8;
        dirptr = dirptr + 32;
        }
}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME:  computeK                       */
/*      PURPOSE:  compute file size                    */
/*      INPUTS:  dirptr                                */
/*      GLOBALS USED:  fileK                           */
/*      GLOBALS CHANGED:  fileK                        */
/*      MODULES CALLED:  compare(asm routine)          */
/*      CALLING MODULES:  nextfile, backup             */
/*******************************************************/


computeK(dirptr)
char *dirptr;
{
char match;
char *filenameptr;      /* pointer to first FCB */

filenameptr = dirptr;

fileK = 0;              /* initialize filesize */

while(peek(dirptr + 1) != 0xE5)
        {
        match = calla(compare,0,filenameptr,12,dirptr);
        if(match)               /* add up size (round up) */
                fileK = fileK + (peek(dirptr + 15) + 7) / 8;
        dirptr = dirptr + 32;                   /* point to next FCB */
        }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  makeDAT                      */
/*      PURPOSE:  make a disk allocation table       */
/*      GLOBALS USED:  DATsize, rec_dir[], d_disk_k,  */
/*                     DAT[]                         */
/*      GLOBALS CHANGED:  DATsize, DAT[]             */
/*      MODULES CALLED:  setDAT                      */
/*      CALLING MODULES:  copy                       */
/*****************************************************/


makeDAT()       /* BUILD DESTINATION DISK ALLOCATION TABLE FROM DIRECTORY */
{
int i,j,k;
int grpnum;
int numdirgrps;         /* number of groups taken up by directory */
int totalgrps;          /* total number of groups DAT can allocate */
int leftover;           /* number of unused groups in last DAT byte */

/* compute # groups in directory */
/* # groups = rec_dir[destin] / 8 rec/grp */

numdirgrps = rec_dir[destin] / 8;

totalgrps = d_disk_k + numdirgrps;

DATsize = totalgrps / 8;                /* one bit for each 1K of disk space */

leftover = 8 -(totalgrps - DATsize * 8);

for(grpnum = 0; grpnum <= DATsize; ++grpnum)
        DAT[grpnum] = 0;                /* initialize DAT a byte at a time */

if(leftover < 8)        /* set unused bits in last DAT byte */
        {
        ++DATsize;      /* compensate for round off */
        for(i = 0; i < leftover; ++i)
                setDAT(totalgrps + i);
        }
else setDAT(totalgrps);

for(grpnum = 0; grpnum < numdirgrps; ++grpnum)
        setDAT(grpnum);    /* set used flags (bits) for grps used by direc */
```

```
/* set DAT bits for all used groups in directory */

for(i = direcbase; peek(i+1) != 0xE5; i = i + 32)

        if(peek(i) != 0xE5)
              for(j = i + 16; j < i + 32; ++j)
                  {
                   k = peek(j);
                   if(k != 0)
                        setDAT(k);
                   else
                        break;
                  }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:                               */
/*      PURPOSE:  Returns the value of the bit which */
/*                corresponds to grpnum. The purpose of */
/*                the bit is to tell whether or not a */
/*                particular group is used (set) or not.*/
/*                This bit is in an array of bytes   */
/*                (DAT[]).                           */
/*      INPUTS:  grpnum                              */
/*      GLOBALS USED:  DAT[]                         */
/*      GLOBALS CHANGED:  DAT[]                      */
/*      CALLING MODULES:  makeDAT                    */
/*****************************************************/


setDAT(grpnum)
int grpnum;
{
char i,bytenum,bitnum;

bytenum = ((grpnum-1) / 8) + 1;
bitnum = ((grpnum-1) % 8) + 1;

for(i = 1; bitnum > 1; i = i*2)
        --bitnum;

DAT[bytenum] = DAT[bytenum] | i;
}
```

```
/****************************************************/
/*      DATE:  18 NOV 83                            */
/*      FUNCTION NAME:  setgrps                     */
/*      PURPOSE:  RUN THRU NEWLY BUILT DAT, AND BUILD  */
/*                ARRAY OF GROUPS THAT CAN BE USED TO  */
/*                HOLD THE NEW FILE. IF NOT ENOUGH     */
/*                GROUPS AVAILABLE THEN PRINT AN ERROR */
/*                MESSAGE                           */
/*      GLOBALS USED:  error, group, DATsize, numfilgrps*/
/*      GLOBALS CHANGED:  error, group              */
/*      CALLING MODULES:  setgrps                   */
/****************************************************/


setgrps()
{
int i,j;
char fits;                      /* flag to show file will fit on new disk */

j = 0;                          /* initialize destination grp number */
fits = 0;                       /* initialize flag to show file fits */

for(i = 1; i <= (DATsize * 8); ++i)

        if(chkDAT(i) == 0)
            {
             group[destin][j] = i;
             setDAT(i);                 /* this grp is now taken */
             ++j;                       /* get ready for next grp number */
             if(j == numfilgrps)
                 {
                  fits = 1;             /* flag to show file fits on new disk */
                  break;                /* finished building array */
                 }
            }

if(! fits)
        error = 1;                      /* global error flag */
else
        error = 0;

}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                            */
/*      FUNCTION NAME: chkDAT                       */
/*      PURPOSE: find if a group is used or not (bit */
/*              set or not)                         */
/*      INPUTS:  grpnum                             */
/*      OUTPUTS:  0 if available else nonzero       */
/*      CALLING MODULES:  setgrps                   */
/*****************************************************/


/* Function to find if a group is used or not (bit set or not) */

chkDAT(grpnum)
int grpnum;
{
char i,bytenum,bitnum;

bytenum = ((grpnum-1) / 8) + 1;
bitnum = ((grpnum-1) % 8) + 1;

for(i = 1; bitnum > 1; i = i*2)          /* set a single bit corresponding */
        --bitnum;                        /* to group number */

return(DAT[bytenum] & i);                /* 0 if available */

}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  readfile                     */
/*      PURPOSE:  load a file from disk to memory    */
/*      INPUTS:  filename, address                   */
/*      CALLING MODULES:  init, getsystems           */
/*****************************************************/


readfile(filename,addr)
char filename[15];
int addr;
{
int fd;                                /* cpm file directory code */

fd = swapin(filename,addr);

if(fd == -1)
         {
          printf("CAN'T FIND %s\n",filename);
          exit();
         }
}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  getdrive                     */
/*      PURPOSE:  input a drive number from user     */
/*      OUTPUTS:  drive                              */
/*      CALLING MODULES:  getsystems                 */
/*****************************************************/


getdrive()
{
char drive;

do
         {
          printf("(A,B,C,D)? \0");
          drive = toupper(getchar());
          printf("\n\n");
         }
while
          ( (drive > 68) || (drive < 65) );       /* ALLOW ONLY A TO D ENTRY */

drive = drive - 65;                          /* CONVERT ASCII TO INT */
     return(drive);
}
```

```
/*******************************************************/
/*      DATE:  18 NOV 83                               */
/*      FUNCTION NAME:  disk_IO                        */
/*      PURPOSE:  call assembly with correct parameters */
/*                to read or write to disk             */
/*      INPUTS:  src_des, rd_wrt, grpnum, numrecs      */
/*      GLOBALS USED:  DTA, offset, rec_trk, rec_sec,  */
/*      GLOBALS CHANGED:  DTA                          */
/*      MODULES CALLED:  systems assembly routines     */
/*      CALLING MODULES:  writedir, getdir, grptxfer   */
/*******************************************************/


disk_IO(src_des,rd_wrt,grpnum,numrecs)
char src_des;                   /* system - source or destination */
char rd_wrt;                    /* read/write flag - 1 for write, else read */
int grpnum;                     /* group number */
int numrecs;                    /* total # records remaining to read */
{
char result;                    /* # of records to transfer */
int noffrec;                    /* record # if there were no offset */
char track;
char sector;                    /* logical sector number */
char numsecs;                   /* number sectors to transfer */

noffrec = (grpnum * 8) + offset[src_des];

track = noffrec / rec_trk[src_des];

sector = ((noffrec % rec_trk[src_des]) + rec_sec[src_des]) / rec_sec[src_des];

numsecs = (numrecs - 1 + rec_sec[src_des]) / rec_sec[src_des];

do
        {
        result = calla(sysaddr[src_des], rd_wrt, DTA, 256*track + sector,
        256*drive[src_des] + numsecs);

        if(result != 0)
                printf("diskIO result is %d\n",result);
        }
while(result != 0);

DTA = DTA + 128 * numrecs;

}
```

```
/*****************************************************/
/*      DATE:  18 NOV 83                             */
/*      FUNCTION NAME:  clearcrt                     */
/*      PURPOSE:  just what it says                  */
/*****************************************************/

clearcrt()
{
putchar(esc);
putchar(clear);
}
```

```
;****************************************************
;*                                                  *
;*      DATE:  28 OCT 83                            *
;*      FILE NAME: UDCLIB                           *
;*      PURPOSE: CONTAINS MANY LIBRARY ROUTINES USED  *
;*               TO INTERFACE TO THE UNIVERSAL FLOPPY *
;*               DISK CONTROLLER.                   *
;****************************************************

LIBADDR   EQU    4000H       ;TRANSIT.C WILL LOAD UDCLIB.THF HERE

;SIO CONSTANTS
SIO_D     EQU    2           ;SIO DATA PORT
SIO_C     EQU    3           ;SIO CONTROL/STATUS PORT
RESET     EQU    18H         ;CODE TO RESET SIO
SYNMODE   EQU    10H         ;CODE FOR SIO SYNCH MODE (16 BIT SYNC CHAR)
TX        EQU    68H         ;CODE TO ENABLE SIO TXMIT
RX        EQU    0D1H        ;CODE TO ENABLE SIO RX
WAITTX    EQU    80H         ;CODE TO TURN ON SIO TX WAIT STATE GENERATOR
WAITRX    EQU    0A0H        ;CODE TO TURN ON SIO RX WAIT STATE GENERATOR
WAITOFF   EQU    0           ;CODE TO TURN OF TX OR RX WAIT STATE GEN

;PIO CONSTANTS
MODE3     EQU    0CFH        ;PIO CONTROL WORD FOR MODE 3 (BIT MODE)
PIOA_D    EQU    4           ;PIO CHANNEL A DATA PORT
PIOB_D    EQU    5           ;PIO CHANNEL B DATA PORT
PIOA_C    EQU    6           ;PIO CHANNEL A CONTROL PORT
PIOB_C    EQU    7           ;PIO CHANNEL B CONTROL PORT

;MFM CONSTANTS
IXAM1     EQU    22H         ;HIGH NIBBLE OF INDEX ADDRESS MARK (A OF A1)
IXAM2     EQU    91H         ;LOW NIBBLE OF INDEX ADDRESS MARK (1 OF A1)
DDAM1     EQU    0AAH        ;DATA ADDRESS MARK (MFM ENCODED F OF FB)
DDAM2     EQU    0A2H        ;DATA ADDRESS MARK (MFM ENCODED B OF FB)
SFMDM1    EQU    0AFH        ;DATA ADDR MARK (FM ENCODED F OF FB) SOFT SEC
SFMDM2    EQU    0F6H        ;DATA ADDR MARK (FM ENCODED B OF FB) SOFT SEC
HFMDM1    EQU    0FFH        ;DATA ADDR MARK (FM ENCODED F OF FB) HARD SEC
HFMDM2    EQU    0F7H        ;DATA ADDR MARK (FM ENCODED B OF FB) HARD SEC
IDFM1     EQU    0AFH        ;ID ADDR MARK (FM ENCODED F OF FE) SOFT SEC
IDFM2     EQU    07EH        ;ID ADDR MARK (FM ENCODED E OF FE) SOFT SEC
IDAM      EQU    0FEH        ;TRACK/SEC ID ADDRESS MARK (NOT ENCODED)
FILL1     EQU    49H         ;FILL BYTES (MFM ENCODED 4 OF 4E)
FILL2     EQU    2AH         ;ENCODED E OF 4E
SIDEFLG   EQU    00          ;SIDE FLAG (0 ALWAYS FOR A SINGLE SIDED DISK)
INITCRC   EQU    0E295H      ;INITIAL CRC VALUE FOR MFM ENCODED DATA
```

```
        .Z80
        ASEG
        ORG     100H            ;SINGLE DENSITY ENCODE TABLE (LOW BYTE...

        .PHASE  LIBADDR
        .RADIX  16              ;OF ADDRESS MUST BE = 00)

SDENTBL: DB     55,0D5,75,0F5,5D,0DD,7D,0FD
        DB      57,0D7,77,0F7,5F,0DF,7F,0FF

DRIVE::   DS    1               ;STORAGE AREA FOR DESIRED DRIVE
HD_SOFT:: DS    1               ;1 IF HARD SECTOR, ELSE SOFT SECTORED
TRACK::   DS    1               ;STORAGE AREA FOR DESIRED TRACK
SECTOR::  DS    1               ;STORAGE AREA FOR DESIRED SECTOR
SEC_TRK:: DS    1               ;SECTORS PER TRACK
BYTSEC::  DS    2               ;STORAGE AREA FOR # BYTES PER SECTOR
DTA::     DS    2               ;POINTER TO DISK TRANSFER ADDRESS
NSECS::   DS    1               ;STORAGE AREA FOR # SECTORS TO TRANSFER
FRSTSEC:: DS    1               ;FIRST SECTOR NUMBER (0 OR 1)
ROUTADR:: DS    2               ;HOLDS POINTER TO USERS SECTOR RD/WRT ROUTINE
SKWFLG::  DS    1               ;SKEW FLAG, 1 IF WANT SKEWING. IF DO, THEN...
SKWTBL::  DS    2               ;ADDRESS OF SKEW TABLE
ERROR::   DS    1               ;ERROR FLAGS (FF-BAD FORMAT, FE-SEEK ERROR)
STKPTR::  DS    2               ;SAVE STACK PTR TO RECOVER FROM NESTED ABORTS

TRKRG0:   DB    0FFH            ;STORAGE AREA FOR HEAD POS OF DRIVE 0 TO 3
TRKRG1:   DB    0FFH            ;0FFH INDICATES THAT THE TRACK REGISTER MAY
TRKRG2:   DB    0FFH            ;DISAGREE WITH ACTUAL HEAD POSITION
TRKRG3:   DB    0FFH            ;
TKROPTR:  DS    2               ;POINTER TO ACTIVE TRK REGISTER
REHMCNT:  DS    1               ;REHOME COUNTER
HANGCNT:  DS    1               ;HANG UP COUNTER
ENCCRC::  DS    4               ;STORAGE FOR ENCODED CRC BYTES
USRMEM:   DS    6               ;GARBAGE MEMORY TO USE
PHYSSEC:  DS    1               ;USED BY FINDSEC TO HOLD PHYSICAL SECTOR #
CRCERRS:  DS    1               ;STORAGE FOR NUMBER OF CRC ERRORS
TBLFLAG:  DB    0FFH            ;FLAG TO INDICATE CRC TABLE HASN'T BEEN LOADED
        .DEPHASE
```

```
        ORG     100H+54H        ;DECODE TABLE FOR BOTH SINGLE AND DOUBLE DEN
        .PHASE  LIBADDR+54H     ;LOW BYTE OF THIS ADDRESS MUST BE AS SHOWN
DECTBL: DB      0,0,80,8,0,0,0,0,40,4,0C0,0C
        .DEPHASE

        ORG     100H+74H
        .PHASE  LIBADDR+74H
        DB      20,2,0A0,0A,0,0,0,0,60,6,0E0,0E
        .DEPHASE

        ORG     100H+0D4H
        .PHASE  LIBADDR+0D4H
        DB      10,1,90,9,0,0,0,0,50,5,0D0,0D
        .DEPHASE


        ORG     100H+0F4H
        .PHASE  LIBADDR+0F4H
        DB      30,3,0B0,0B,0,0,0,0,70,7,0F0,0F
        .DEPHASE

        ORG     100H+100H       ;DOUBLE DENSITY ENCODE TABLE (MUST START...
                                ;WITH ADDRESS OF LOWEST BYTE = 00)
        .PHASE  LIBADDR+100H
DDENTBL: DB     55,95,25,0A5,49,89,29,0A9,52,92,22,0A2,4A,8A,2A,0AA
        DB      54,94,24,0A4,48,88,28,0A8,52,92,22,0A2,4A,8A,2A,0AA

        .RADIX  10
DRVTBL: DB      11101111B,11011111B,10111111B,01111111B

        ;SKEW TABLE FOR IBM STANDARD 8" SINGLE DENSITY
IBMSKW:: DB     1,7,13,19,25,5,11,17,23,3,9,15,21
        DB      2,8,14,20,26,6,12,18,24,4,10,16,22

IDCRCS: DS      2               ;PUT TRACK/SECTOR ID CRCS HERE
CRCPTR:: DS     256             ;STORAGE AREA FOR CRC VALUES
CRCTBL: DS      512             ;STORAGE AREA FOR CRC TABLE
```

```
;****************************** COMPARE **************************************
;COMPARE IS USED BY TRANSIT TO VERIFY A FILE

COMPARE:: LD      A,(DE)
          INC     DE
          CPI                     ;CHECK ONE BYTE FOR A MATCH
          JR      NZ,COMPERR
          JP      PE,COMPARE      ;CONTINUE UNTIL BC GOES TO ZERO
          LD      A,1             ;ONE IN ACCUM MEANS MATCHED
          RET

COMPERR:  XOR     A               ;ZERO IN ACCUM MEANS ERROR
          RET

;************************ PIO INITIALIZATION *******************************
;INIT PIO AND GET DRIVE MOTORS UP TO SPEED

PIOINIT:: LD      A,MODE3         ;SET CHAN A INTO MODE 3 (BIT MODE)
          OUT     (PIOA_C),A
          LD      A,00000001B     ;DEFINE BITS 1-7 OUTPUTS, BIT 0 INPUT (CHAN A)
          OUT     (PIOA_C),A
          LD      A,MODE3         ;SET CHAN B INTO MODE 3 (BIT MODE)
          OUT     (PIOB_C),A
          LD      A,00000011B     ;DEFINE BITS 2-7 OUTPUTS, BIT 0-1 INPUT (B)
          OUT     (PIOB_C),A
          LD      A,00000100B     ;DEACTIVATE WRITE GATE
          OUT     (PIOB_D),A
          LD      A,11110110B     ;DESELECT ALL DRIVERS, TURN ON MOTORS
          OUT     (PIOA_D),A
          LD      BC,5000         ;DELAY .5 SEC FOR DRIVES TO GET UP TO SPEED
          CALL    DELAY
          RET

;****************************** DESELECT ************************************
DSELECT:: IN      A,(PIOA_D)
          OR      0F0H            ;DESELECT ALL DRIVES
          OUT     (PIOA_D),A
          RET
```

```
;************************* PUSH RETURN ADDRESS **********************************
;SUBROUTINE USED TO ALLOW A CALREL MACRO INSTRUCTION (CALL RELATIVE)
;THIS ROUTINE PUTS THE CORRECT RETURN ADDRESS INTO THE STACK

PSHRET::  EXX                    ;SAVE ALL REGISTERS
          POP     HL             ;GET RET ADDRESS
          LD      B,H            ;SAVE IT
          LD      C,L
          INC     HL             ;COMPUTE RET ADDRESS FOR CALREL MACRO
          INC     HL
          PUSH    HL             ;PUT CALREL RET ADDRESS IN STACK
          PUSH    BC             ;RESTORE RETURN ADDRESS TO CALLING ROUTINE
          EXX                    ;RESTORE REGISTERS
          RET
```

```
;*********************** SEQUENTIAL SECTOR TRANSFER ***********************
;TRANSFERS AN UNLIMITED NUMBER OF CONTIGUOUS SECTORS (READ OR WRITE)
;THE FOLLOWING MEMORY POSITIONS MUST BE SET:
;
;IF HARD SECTORED DISK SET (HD_SOFT) TO 1, ELSE SET IT TO SOMETHING ELSE
;START SECTOR IS (SECTOR)
;START TRACK IS (TRACK)
;NUMBER OF SECTORS IS (NSECS)
;SECTOR TRANSFER ROUTINE IS (ROUTADDR)
;SECTORS PER TRACK IS (SEC_TRK)
;
;IX IS USED AS POINTER TO CRC BYTES BY LIBRARY SECTOR READ AND WRITE...
;ROUTINES, CRC BYTES DO NOT HAVE TO BE THERE UNLESS THE LIBRARY READ SECTOR...
;AND WRITE SECTOR ROUTINES ARE USED

TXFER::  LD      IX,CRCPTR       ;POINT TO FIRST BYTE OF FIRST CRC OF FIRST SEC

TXFLOOP: LD      A,(NSECS)
         CP      0               ;RET IF NO SECTORS REMAINING TO BE TXFERED
         RET     Z               ;DONE
         CALL    SEEK            ;PUT HEAD OVER DESIRED TRACK
         LD      A,(SECTOR)
         LD      E,A
         LD      A,(SEC_TRK)     ;SECTORS PER TRACK
         SUB     E
         INC     A
         LD      E,A             ;E = NUM SECS TO END OF TRACK FROM (SECTOR)
         LD      A,(NSECS)
         CP      E               ;C SET IF NSECS > NUM SECS TO END OF TRK
         JR      NC,TXFER2       ;SKIP ADJUSTMENT OF #SECS TO TXFER
         LD      E,A             ;E = NUM SECS TO READ FROM CURRENT TRACK

TXFER2:  SUB     E               ;COMPUTE NUM SECS REMAINING AFTER THIS TXFER
         LD      (NSECS),A
         CALL    TXFTRK          ;GO TXFER ALL SECTORS OF ONE TRACK
         LD      A,(FRSTSEC)
         LD      (SECTOR),A      ;IF TRANSFER DATA AGAIN,IT WILL BE SECTOR...
         LD      HL,TRACK        ;FIRST SECTOR OF NEXT TRACK
         INC     (HL)            ;
         JR      TXFLOOP         ;TRANSFER MORE TRACKS IF ANY REMAINING
```

```
;ROUTINE TO TXFER CONTIGUOUS SECTORS ON ONE TRACK STARTING WITH...
;THE SECTOR NUMBER IN (SECTOR)
;NUMBER OF SECTORS TO TXFER IS IN E
;
TXFTRK:   LD      A,(HD_SOFT)     ;IF HAVE A HARD SECTORED DISK, THEN WAIT...
          DEC     A               ;FOR CORRECT SECTOR PULSE BEFORE BRANCHING
          CALL    Z,FNDHSEC
TRKLOOP:  PUSH    DE              ;SAVE SECTOR COUNTER
          LD      HL,RETHERE      ;PUT RETURN ADDRESS IN STACK
          PUSH    HL
          LD      HL,(ROUTADR)    ;GET ADDRESS OF USERS SECTOR RD/WRT ROUTINE
          JP      (HL)

RETHERE:  POP     DE              ;RESTORE SECTOR COUNTER
          LD      HL,SECTOR
          INC     (HL)            ;INCREMENT SECTOR NUMBER
          DEC     E               ;SEE IF DONE YET
          JR      NZ,TRKLOOP
          RET
```

```
;************************** DRIVE CONTROL ************************************
;
DRVSET::  PUSH    AF              ;SAVE SIZE & DENSITY BITS
          LD      HL,DRVTBL
          LD      A,(DRIVE)       ;GET THE DESIRED DRIVE (0-3)
          LD      C,A
          LD      B,0             ;COMPUTE DRIVE TABLE ADDRESS
          ADD     HL,BC
          LD      D,(HL)          ;GET BITS TO SELECT DESIRED DRIVE
          IN      A,(PIOA_D)      ;GET CURRENT DATA IN PIO CHAN A
          OR      0F0H            ;BITS TO DESELECT ALL DRIVES
          AND     D               ;RESET BIT FOR DESIRED DRIVE (ACTIVE LOW)
          OUT     (PIOA_D),A      ;SELECT THE DRIVE
          POP     AF              ;PUT SIZE & DENSITY BITS IN A
          OR      00000100B       ;LEAVE WRITE GATE OFF (BIT 2 - ACTIVE LOW)
          OUT     (PIOB_D),A      ;SEND SIZE AND DENSITY TO CONTROLLER

          ;SET TRACK REGISTER POINTER
          LD      HL,TRKRG0
          ADD     HL,BC
          LD      (TKRGPTR),HL    ;TKRGPTR CONTAINS THE ADDRESS OF THE TRACK
                                  ;REGISTER FOR THE ACTIVE DRIVE
          ;DELAY TO ENSURE DISK IS SPINNING AND UP TO SPEED
          CALL    IXPLS
          CALL    IXPLS
          RET
;*** HOME ***
;MOVE HEAD OF ACTIVE DRIVE TO TRK 0
;SET ACTIVE TRACK REGISTER TO ZERO

HOME::    IN      A,(PIOA_D)
          AND     1               ;IF BIT 0 IS SET THEN NOT AT TRACK 0 YET
          JR      Z,HOME2         ;DONE, HEAD IS POSITIONED OVER TRACK 0
          CALL    STPOUT          ;GO STEP OUT ONE TRACK (TRK 0 IS OUTER MOST)
          JR      HOME            ;GO CHECK AGAIN TO SEE IF AT TRK 0
HOME2:    LD      HL,(TKRGPTR)    ;PUT ADDRESS OF ACTIVE TRK REGISTER IN HL
          XOR     A               ;ZERO OUT TRACK REGISTER
          LD      (HL),A          ;TRACK# AND TRACK REGISTER ARE NOW ZERO
          RET
```

```
;*** SEEK ***
;MOVE HEAD TO TRACK SPECIFIED BY (TRACK)

SEEK::   LD      HL,(TKRGPTR)   ;PUT ADDRESS OF ACTIVE TRK REGISTER IN HL
         LD      A,(HL)         ;GET CURRENT POSITION OF HEAD
         CP      0FFH           ;FLAG TO SHOW NEVER BEEN SET
         JR      NZ,SEEK2
         CALL    HOME           ;GO TO A KNOWN HEAD POSITION

SEEK2:   LD      B,A            ;CURRENT HEAD POSITION
         LD      A,(TRACK)      ;GET DESIRED TRACK NUMBER
         LD      (HL),A         ;UPDATE TRACK REGISTER (IT'S NOW = DESIRED TK)
         SUB     B              ;COMPUTE DIFF BETWEEN DESIRED AND ACTUAL TRK
         RET     Z              ;DESIRED TRACK IS ALREADY = ACTUAL TRACK
         JR      C,SKOUT        ;GO SEEK OUTWARD IF CARRY, ELSE SEEK INWARD

;*** HEAD CONTROL ***
;
SK_IN:   LD      D,A            ;NUMBER OF STEPS INWARD NEEDED
SKINLP:  CALL    STPIN          ;STEP IN ONE TRK
         DEC     D              ;SEE IF TIME TO STOP STEPPING
         JR      NZ,SKINLP      ;CONTINUE STEPPING UNTIL D GOES TO ZERO
         RET                    ;HEAD IS NOW OVER DESIRED TRACK

SKOUT:   CPL                    ;COMPUTE ABSOLUTE VAL OF DIFFERENCE
         INC     A              ;NOW HAVE NUMBER OF STEPS NEEDED
         LD      D,A
SKOUTLP: CALL    STPOUT         ;STEP OUT ONE TRK
         DEC     D              ;SEE IF TIME TO STOP STEPPING
         JR      NZ,SKOUTLP     ;
         RET                    ;HEAD IS NOW OVER DESIRED TRACK

STPOUT:  IN      A,(PIOA_D)     ;GET CURRENT DRIVE COMMAND BYTE
         OR      00000100B      ;BIT 2 SET MEANS DIRECTION IS OUT
         OUT     (PIOA_D),A     ;SET BIT 2
         CALL    PULSE          ;GO GIVE THE STEPPING PULSE
         RET
```

```
STPIN:   IN     A,(PIOA_D)      ;GET CURRENT DRIVE COMMAND BYTE
         AND    11111011B       ;BIT 2 RESET MEANS DIRECTION IS IN
         OUT    (PIOA_D),A      ;RESET BIT 2
         CALL   PULSE           ;GO GIVE THE STEP PULSE
         RET

PULSE:   PUSH   AF              ;SAVE TO RETURN STEP BIT TO ITS NORMAL HIGH
         AND    11111101B       ;BRING BIT 1 LOW (STEP BIT)
         OUT    (PIOA_D),A
         LD     BC,1            ;LEAVE PULSE LOW FOR .1 MS
         CALL   DELAY           ;
         POP    AF              ;BRING STEP BIT BACK TO IT'S NORMAL HIGH
         OR     00000010B       ;INSURE STEP BIT IS HIGH
         OUT    (PIOA_D),A
         LD     BC,200          ;LEAVE PULSE HIGH FOR 20MS (15 MS MINIMUM)
         CALL   DELAY           ;BEFORE ALLOWING ANOTHER PULSE TO BE GENERATED
         RET

DELAY::  LD     A,24            ;CONSTANT TO MAKE SHTLP DELAY .1 MS
SHTLP:   DEC    A               ;SHORT LOOP (.1 MILLISECOND)
         JR     NZ,SHTLP

         DEC    BC              ;DELAY .1 MS * (ORIGINAL BC VALUE)
         LD     A,B             ;SEE IF BC IS ZERO YET
         OR     C
         JR     NZ,DELAY
         RET
```

```
;****************************** DISK READ *************************************
;IX MUST POINT TO MEMORY TO RECEIVE CRC BYTES

;*** READ SOFT SECTORED FM ***

RDSFM::   CALL    FNDFSEC         ;WAIT FOR CORRECT SECTOR TO SHOW UP
          LD      D,SFMDM1        ;INIT SIO WITH DATA ADDR MARK AS SYNCH CHAR
          LD      E,SFMDM2
          CALL    SIOINIT
          CALL    GENREAD
          RET

;*** READ HARD SECTORED FM ***

RDHDFM::  CALL    HIGHP           ;WAIT FOR SECTOR PULSE TO GO HIGH
          LD      DE,05555H       ;INIT SIO WITH DATA ZERO AS SYNCH CHAR
          CALL    SIOINIT
          CALL    RXENABL
          IN      A,(SIO_D)       ;PUT THE CPU IN A WAIT STATE TILL
                                  ;SYNCH CHAR IS FOUND
          CALL    STOP            ;TURN OFF WAIT STATE GENERATOR

          LD      D,HFMDM1        ;FM ENCODED DATA ADDRESS MARK FOR HARD SECTOR
          LD      E,HFMDM2
          CALL    SIOINIT
          CALL    GENREAD         ;GENERAL READ ROUTINE
          RET
;*** READ MFM SECTOR ***
;IX MUST POINT TO MEM LOCATION TO RECEIVE CRC BYTES
RDMFM::   CALL    FNDMSEC         ;WAIT FOR CORRECT SECTOR TO SHOW UP
          CALL    PSTFILR         ;WAIT TILL FILL BYTES PASS
          LD      D,DDAM1         ;INIT SIO WITH DATA ADDR MARK AS SYNCH CHAR
          LD      E,DDAM2
          CALL    SIOINIT
          CALL    GENREAD         ;GENERAL READ ROUTINE
          RET
```

```
;*** GENERAL READ ***
GENREAD: LD     BC,(BYTSEC)     ;NUMBER OF DATA BYTES TO READ
         LD     HL,DECTBL       ;SET H TO DECODE TABLE TOP ADDRESS
         LD     DE,(DTA)        ;PUT THE BYTES AT THE DISK TXFER ADDRESS
         CALL   RXENABL         ;CRANK UP THE SIO RECIEVER
         CALL   RCEIV           ;BRING THE BYTES INTO MEMORY
         CALL   STOP            ;DISABLE WAIT STATE GENERATOR
         LD     (DTA),DE
         RET

RCEIV::  IN     A,(SIO_D)       ;GET HIGH ORDER NIBBLE OF DATA W/ CLK BITS
         OR     01010101B       ;SET CLK BITS SO CAN USE SAME TABLE AS S_DEN
         DEC    A               ;SET TABLE ADDRESS FOR HIGH NIBBLE...
         LD     L,A             ;
         LD     A,(HL)          ;GET DECODED NIBBLE
         EX     AF,AF'          ;SAVE IT
         IN     A,(SIO_D)       ;GET LOW ORDER NIBBLE OF DATA W/ CLK BITS
         OR     01010101B       ;SET CLK BITS SO AN USE SAME TABLE AS S_DEN
         LD     L,A             ;SET TABLE ADDRESS FOR LOW NIBBLE
         EX     AF,AF'          ;RESTORE HIGH NIBBLE
         OR     (HL)            ;PUT NIBBLES TOGETHER TO FORM THE BYTE OF DATA
         LD     (DE),A          ;SAVE ONE BYTE OF DATA
         INC    DE              ;POINT TO MEM WHICH RECEIVES NEXT BYTE
         DEC    BC              ;SEE IF DONE YET
         LD     A,B
         OR     C
         JR     NZ,RCEIV

         ;GET CRC BYTES
         LD     B,2             ;2 CRC BYTES
CRCIN:   IN     A,(SIO_D)       ;GET HIGH ORDER NIBBLE WITH CLK BITS
         OR     01010101B       ;SET CLK BITS SO CAN USE SAME TABLE AS S_DEN
         DEC    A               ;SET TABLE ADDRESS FOR HIGH NIBBLE...
         LD     L,A             ;
         LD     A,(HL)          ;GET DECODED NIBBLE
         EX     AF,AF'          ;SAVE IT
         IN     A,(SIO_D)       ;GET LOW ORDER NIBBLE OF DATA W/ CLK BITS
         OR     01010101B       ;SET CLK BITS SO AN USE SAME TABLE AS S_DEN
```

```
        LD      L,A             ;SET TABLE ADDRESS FOR LOW NIBBLE
        EX      AF,AF'          ;RESTORE HIGH NIBBLE
        OR      (HL)            ;PUT NIBBLES TOGETHER TO FORM THE BYTE
        LD      (IX+0),A        ;SAVE ONE CRC BYTE
        INC     IX              ;POINT TO MEM WHICH RECEIVES NEXT CRC BYTE
        DJNZ    CRCIN
        RET

;************************* WRITE TO DISK *************************************
;IX MUST POINT TO STRING OF CRC BYTES

;*** WRITE SOFT SECTORED FM ***
;
WRTSFM:: ;SET UP ALT REGS FOR FAST CRC WRITE
        EXX
        LD      B,2             ;2 CRC BYTES
        LD      DE,SDENTBL      ;SINGLE DENSITY ENCODE TABLE
        PUSH    IX              ;PUT CRC POINTER IN HL
        POP     HL
        EXX

        CALL    FNDFSEC         ;WAIT FOR SECTOR TO SHOW UP
        LD      DE,0FFFFH       ;FILL BYTES PRIOR TO DATA
        CALL    SIOINIT
        LD      BC,(BYTSEC)     ;BYTES PER SECTOR
        LD      DE,SDENTBL      ;SINGLE DENSITY ENCODE TABLE
        LD      HL,(DTA)        ;DISK TRANSFER ADDRESS
        CALL    TXENABL         ;CRANK UP THE SIO TRANSMITTER

        IN      A,(PIOB_D)      ;TURN ON THE WRITE GATE...
        PUSH    AF              ;SAVE SO CAN RETURN TO ORIG STATE LATER
        AND     11111011B       ;WRITE GATE ACTIVE LOW
        OUT     (PIOB_D),A      ;DATE IS NOW BEING WRITTEN TO DISK
```

```
        ;WRITE 7 FF'S
        LD      A,0FFH          ;NIBBLE OF FF WITH FM ENCODING
        LD      E,14            ;14 NIBBLES = 7 BYTES
FFLOOP: OUT     (SIO_D),A       ;WRITE ONE NIBBLE
        DEC     E
        JR      NZ,FFLOOP

        ;WRITE 6 ZEROS
        LD      A,55H           ;NIBBLE OF 00 WITH FM ENCODING
        LD      E,12            ;12 NIBBLES = 6 BYTES
FMZLP:  OUT     (SIO_D),A       ;WRITE ONE NIBBLE
        DEC     E
        JR      NZ,FMZLP

        ;WRITE SOFT SECTORED FM DATA ADDRESS MARK
        LD      A,SFMDM1
        OUT     (SIO_D),A
        LD      A,SFMDM2
        OUT     (SIO_D),A

        ;WRITE DATA AND CRC BYTES
        CALL    WFMLOOP

        ;CLEAN UP
        CALL    STOP            ;TURN OFF WAIT STATE GENERATOR
        POP     AF              ;RESTORE ORIGINAL DRIVE STATE...
        OUT     (PIOB_D),A      ;TURN OFF WRITE GATE
        LD      (DTA),HL        ;UPDATE DISK TRANSFER ADDRESS
        RET
```

```
;*** WRITE HARD SECTORED FM ***
;
WRHDFM:: LD     DE,5555H        ;INIT SIO WITH DATA 0 WITH FM CLOCK BITS AS...
         CALL   SIOINIT         ;SYNCH CHAR

         EXX                    ;SET UP ALT REGISTERS FOR A FAST CRC WRITE
         LD     B,2             ;2 CRC BYTES
         LD     DE,SDENTBL      ;FM (SINGLE DENSITY) ENCODE TABLE ADDRESS
         PUSH   IX              ;POINTER TO CRC BYTES
         POP    HL
         EXX

         LD     BC,(BYTSEC)     ;BYTES PER SECTOR
         LD     DE,SDENTBL      ;SINGLE DENSITY (FM) ENCODE TABLE ADDRESS
         LD     HL,(DTA)        ;DISK TRANSFER ADDRESS
         CALL   TXENABL         ;CRANK UP THE SIO TRANSMITTER

         CALL   HIGHP           ;WAIT FOR SECTOR TO ARRIVE
         IN     A,(PIOB_D)      ;TURN ON WRITE GATE
         PUSH   AF              ;SAVE SO CAN RETURN TO SAME STATE
         AND    11111011B       ;BIT 2 LOW = WRITE GATE ON
         OUT    (PIOB_D),A      ;GATE'S ON, DATA IS NOW BEING WRITTEN

         ;WRITE 16 BYTES OF ZEROS
         LD     A,55H           ;NIBBLE OF DATA ZERO WITH '1' CLOCK BITS
         LD     E,32            ;SEND 32 NIBBLES / 16 BYTES OF ZEROS
FMZRLP:  OUT    (SIO_D),A       ;WRITE TO DISK
         DEC    E
         JR     NZ,FMZRLP       ;REPEAT TILL ALL 16 DATA ZEROS ARE SENT

         ;SEND DATA ADDRESS MARK
         LD     A,HFMDM1        ;FM ENCODED HIGH NIBBLE OF ADDRESS MARK
         OUT    (SIO_D),A
         LD     A,HFMDM2        ;FM ENCODED LOW NIBBLE OF ADDRESS MARK
         OUT    (SIO_D),A

         ;WRITE DATA AND CRC BYTES
         CALL   WFMLOOP
```

```
        ;CLEAN UP
        CALL    STOP            ;DISABLE WAIT STATE GENERATOR
        POP     AF              ;GET ORIGINAL DRIVE STATE (BEFORE WRITE)
        OUT     (PIOB_D),A      ;TURN OFF WRITE GATE
        LD      (DTA),HL        ;PREP FOR FOLLOWING SECTOR WRITE
        RET

;*** SEND FM ENCODED DATA TO DISK ***
;
WFMLOOP: LD     A,(HL)          ;FIRST BYTE TO TRANSMIT
        RRCA                    ;GET HIGH NIBBLE OF BYTE TO ENCODE AND WRITE
        RRCA
        RRCA
        RRCA
        AND     0FH             ;MASK LOW NIBBLE
        LD      E,A
        LD      A,(DE)          ;ENCODED HIGH NIBBLE FROM TABLE
        OUT     (SIO_D),A       ;SEND IT TO DISK
        LD      A,(HL)          ;GET LOW NIBBLE OF BYTE TO ENCODE AND WRITE
        AND     0FH             ;MASK HIGH NIBBLE
        LD      E,A
        LD      A,(DE)          ;ENCODED LOW NIBBLE FROM TABLE
        INC     HL              ;POINT TO NEXT BYTE TO SEND
        DEC     BC              ;BYTE COUNTER
        OUT     (SIO_D),A       ;SEND IT TO DISK
        LD      A,B             ;SEE IF DONE YET
        OR      C
        JR      NZ,WFMLOOP
```

```
          ;WRITE CRC BYTES
          EXX                         ;GET REGISTERS PREPARED FOR CRC WRITE
FMCRC:    XOR     A
          RLD                         ;GET HIGH NIBBLE OF BYTE TO ENCODE AND WRITE
          LD      E,A
          LD      A,(DE)              ;ENCODED HIGH NIBBLE FROM TABLE
          OUT     (SIO_D),A           ;SEND IT TO DISK
          XOR     A
          RLD                         ;GET LOW NIBBLE OF BYTE TO ENCODE AND WRITE
          LD      E,A
          LD      A,(DE)              ;ENCODED LOW NIBBLE FROM TABLE
          OUT     (SIO_D),A           ;SEND IT TO DISK
          INC     HL                  ;POINT TO NEXT BYTE TO SEND
          DJNZ    FMCRC

          PUSH    HL                  ;UPDATE IX REG (POINTER TO CRC)
          POP     IX

          EXX                         ;RESTORE ORIGINAL REGISTERS
          OUT     (SIO_D),A           ;PUT SIO IN ONE MORE WAIT STATE TO GET LAST...
          RET                         ;CRC BYTE OUT BEFORE WRITE GATE IS TURNED OFF

;*** WRITE AN MFM SECTOR (<= 256 BYTES) ***
;
WRTMFM::  CALL    CRCENCOD            ;ENCODE CRC BYTES FOR THIS SECTOR
          CALL    FNDMSEC             ;WAIT FOR DESIRED SECTOR TO SHOW UP
          CALL    PSTFILR             ;WAIT FOR FILL BYTES TO PASS
          LD      DE,5555H            ;START UP SIO TXMITER WITH DATA ZERO WITH
          CALL    SIOINIT             ;ITS CLOCK BITS AS A SYNC CHAR...
          CALL    TXENABL             ;
          IN      A,(PIOB_D)          ;TURN ON WRITE GATE
          PUSH    AF
          AND     11111011B
          OUT     (PIOB_D),A          ;GATE'S ON, DATA IS NOW BEING WRITTEN
```

```
         LD      BC,(BYTSEC)     ;NUMBER OF BYTES TO WRITE
         LD      A,B             ;WMFMLOOP DOES NOT HAVE TIME TO DEC A REG...
         CP      0               ;PAIR, THEREFORE DJNZ IS USED, WHICH NEEDS...
         JR      NZ,SKIPINC      ;LOW ORDER BYTE IN B
         INC     A
SKIPINC: LD      B,C
         LD      C,A
         LD      DE,DDENTBL      ;DOUBLE DENSITY ENCODE TABLE
         LD      HL,(DTA)        ;GET DATA BYTES FROM HERE
         CALL    TXMIT           ;SEND BYTES TO DISK
         CALL    STOP            ;DISABLE WAIT STATE GENERATOR
         POP     AF              ;GET ORIGINAL DRIVE STATE (BEFORE WRITE)
         OUT     (PIOB_D),A      ;TURN OFF WRITE GATE
         LD      (DTA),HL        ;UPDATE DISK TRANSFER ADDRESS
         RET

         ;SEND STRING OF ZEROS PRIOR TO DATA AM AND DATA
TXMIT:   LD      A,055H          ;NIBBLE OF DATA ZERO WITH '1' CLOCK BITS
         LD      E,24            ;SEND 24 NIBBLES / 12 BYTES OF ZEROS
ZLOOP:   OUT     (SIO_D),A       ;WRITE TO DISK
         DEC     E
         JR      NZ,ZLOOP        ;REPEAT TILL ALL 12 DATA ZEROS ARE SENT

         ;SEND 3 ADDRESS MARKS
         LD      E,3
ADMRKLP: LD      A,IXAM1         ;ENCODED HIGH NIBBLE OF INDEX ADDRESS MARK
         OUT     (SIO_D),A
         LD      A,IXAM2         ;ENCODED LOW NIBBLE OF INDEX ADDRESS MARK
         OUT     (SIO_D),A
         DEC     E
         JR      NZ,ADMRKLP

         ;SEND DATA ADDRESS MARK
         LD      A,DDAM1         ;GET ENCODED NIBBLE OF DATA ADDRESS MARK
         OUT     (SIO_D),A       ;WRITE AM TO DISK
         LD      A,DDAM2
         SCF                     ;SET UP EARLY FOR WDLOOP SO DON'T MISS A BYTE
         PUSH    AF              ;CF HOLDS LAST DATA BIT OUT THE SIO, NEEDED...
         OUT     (SIO_D),A       ;TO GET THE DD CLOCK ENCODING CORRECT
```

```
        ;SEND DATA ADDRESSED BY HL ENCODED IN MFM
WMFMLP: POP     AF              ;GET LAST BIT SENT FROM SIO VIA CARRY
        LD      A,(HL)          ;GET NEW BYTE TO TRANSMIT
        RRA                     ;INSTALL PREVIOUS BIT TO GET ENCODING CORRECT
        PUSH    AF              ;SAVE CARRY FOR NEXT LOOP
        RRA                     ;SET TABLE ADDRESS FOR HIGH NIBBLE ONLY
        RRA
        RRA
        AND     00011111B
        LD      E,A
        LD      A,(DE)          ;GET ENCODED BYTE TO SEND
        OUT     (SIO_D),A       ;SEND BYTE TO DISK VIA SIO

        LD      A,(HL)          ;GET DATA
        INC     HL              ;GET READY FOR FOLLOWING LOOP
        AND     00011111B
        LD      E,A             ;SET ENCODE TABLE ADDRESS
        LD      A,(DE)          ;GET ENCODED BYTE TO SEND
        OUT     (SIO_D),A       ;SEND ENCODED LOW NIBBLE

        DJNZ    WMFMLP
        DEC     C               ;SEE IF NEED ANOTHER 256 BYTES SENT
        JR      NZ,WMFMLP
        POP     AF              ;CLEAN OUT STACK

        ;SEND CRC BYTES
        LD      A,(IY+0)        ;HIGH ORDER NIBBLE OF FIRST CRC BYTE
        OUT     (SIO_D),A
        LD      A,(IY+1)        ;LOW ORDER NIBBLE OF FIRST CRC BYTE
        OUT     (SIO_D),A
        LD      A,(IY+2)        ;HIGH ORDER NIBBLE OF SECOND CRC BYTE
        OUT     (SIO_D),A
        LD      A,(IY+3)        ;LOW ORDER NIBBLE OF SECOND CRC BYTE
        OUT     (SIO_D),A
```

```
        ;SEND 4E AFTER CRC BYTE
        RLA                             ;GET LAST DATA BIT OF CRC
        CCF                             ;NEXT CLOCK BIT OUT IS COMPLEMENT OF LAST...
        RLA                             ;...DATA BIT OUT
        AND     00000001B
        OR      48H                     ;ENCODED '4' OF 4E
        OUT     (SIO_D),A
        LD      A,2AH                   ;ENCODED 'E' OF 4E
        OUT     (SIO_D),A

        OUT     (SIO_D),A               ;PUT SIO IN ONE LAST WAIT STATE TO ALLOW...
                                        ;PREVIOUS BYTE TO BE TXMITTED BEFORE...
                                        ;TURNING OFF THE WRITE GATE
        RET


;************************* FIND MFM SECTOR ***********************************
;THIS ROUTINE WAITS TILL THE DESIGNATED SECTOR SHOWS UP BEFORE RETURNING
;IF AN ERROR IS FOUND IN THE TRACK ID, UP TO 8 REHOMES WILL BE ATTEMPTED,
;IF AN ERROR STILL EXISTS, A JUMP TO ABORT WILL OCCUR, & (ERROR) SET TO FFH
;IF SKEW FLAG IS SET THEN A TABLE ADDRESSED BY (SKWTBL) WILL BE USED TO
;CONVERT THE SECTOR NUMBER

FNDMSEC:: LD    A,(SECTOR)
        LD      E,A
        LD      A,(SKWFLG)              ;SEE IF SKEW IS WANTED
        DEC     A
        CALL    Z,COMPSKW               ;CONVERT LOGICAL TO PHYSICAL SECTOR NUMBER

        LD      A,E                     ;PHYSICAL SECTOR NUMBER
        LD      (PHYSSEC),A
        LD      A,8                     ;INITIALIZE REHOME COUNTER
        LD      (REHMCNT),A
        LD      A,52                    ;INITIALIZE HANG-UP COUNTER
        LD      (HANGCNT),A
```

```
FNDSC2: LD      HL,HANGCNT      ;DECREMENT HANG-UP COUNTER
        DEC     (HL)
        CALL    Z,HANGUP

        CALL    PSTFILR         ;WAIT FOR FILL BYTES TO PASS
        LD      D,IXAM1         ;INIT SIO WITH ADDRESS MARK AS SYNCH CHAR
        LD      E,IXAM2
        CALL    SIOINIT
        CALL    RXENABL
        LD      B,2             ;LOOK FOR 2 ADDR MARKS AFTER INITIAL ONE
        LD      HL,DECTBL       ;SET H TO DECODE TABLE TOP ADDRESS
        LD      DE,USRMEM       ;PUT ID BYTES IN THIS TEMP USER MEMORY

AMLOOP: IN      A,(SIO_D)       ;INPUT A BYTE AFTER INITIAL ADDRESS MARK
        CP      IXAM1           ;SHOULD BE ANOTHER ADDRESS MARK
        JR      NZ,FNDSC2       ;ITS NOT, START OVER
        IN      A,(SIO_D)
        CP      IXAM2           ;LOW ORDER NIBBLE OF ADDRESS MARK
        JR      NZ,FNDSC2       ;ITS NOT, START OVER
        DJNZ    AMLOOP          ;LOOK FOR TWO ADDRESS MARK AFTER INITIAL ONE

        LD      BC,5            ;NUMBER BYTES TO READ (FE, TRK, 00, SEC, 00)
        PUSH    IX              ;PRESERVE IX FOR DATA CRC BYTES
        LD      IX,IDCRCS
        CALL    RCEIV           ;BRING THE BYTES INTO MEMORY
        CALL    STOP            ;DISABLE WAIT STATE GENERATOR
        POP     IX              ;RESTORE DATA CRC BYTE POINTER
```

```
		;SEE IF ID IS CORRECT
IDCHK:	LD	HL,USRMEM	;POINT TO WHAT SHOULD BE ID ADDRESS MARK (FE)
	LD	A,(HL)
	CP	IDAM		;SEE IF IT IS THE ID ADDRESS MARK
	JR	NZ,FNDSC2	;IT'S NOT, START OVER
	INC	HL
	LD	A,(TRACK)
	CP	(HL)		;SEE IF TRACK MATCHES
	JR	NZ,FNDSC2	;IT DOESN'T MATCH, START OVER
	INC	HL
	LD	A,(HL)		;GET WHAT SHOULD BE THE SIDE FLAG
	CP	SIDEFLG		;
	JR	NZ,FNDSC2
	INC	HL
	LD	A,(PHYSSEC)	;GET PHYSICAL SECTOR NUMBER
	CP	(HL)		;SEE IF MATCHES WHAT'S ON DISK
	JR	NZ,FNDSC2	;DOESN'T MATCH
	RET			;DAA, DAA, FOUND IT!

;************************** FIND FM SECTOR *********************************
;THIS ROUTINE WAITS TILL THE DESIGNATED SECTOR SHOWS UP BEFORE RETURNING
;IF AN ERROR IS FOUND IN THE TRACK ID, UP TO 8 REHOMES WILL BE ATTEMPTED,
;IF AN ERROR STILL EXISTS, A JUMP TO ABORT WILL OCCUR AND (ERROR) SET TO FFH
;IF SKEW FLAG IS SET THEN A TABLE ADDRESSED BY (SKWTBL) WILL BE USED TO
;CONVERT THE SECTOR NUMBER

FNDFSEC:: LD	A,(SECTOR)
	LD	E,A
	LD	A,(SKWFLG)	;SEE IF SKEW IS WANTED
	DEC	A
	CALL	Z,COMPSKW	;CONVERT LOGICAL SECTOR # TO PHYSICAL SECTOR #
	LD	A,E		;PHYSICAL SECTOR NUMBER
	LD	(PHYSSEC),A
	LD	A,8		;INITIALIZE REHOME COUNTER
	LD	(REHMCNT),A
	LD	A,52		;INITIALIZE HANG-UP COUNTER
	LD	(HANGCNT),A
```

```
FNDFSC2: LD      HL,HANGCNT      ;DECREMENT HANG-UP COUNTER
         DEC     (HL)
         CALL    Z,HANGUP        ;GO SEE IF IT'S TIME FOR A REHOME

         ;READ IN ID FIELD
         LD      D,IDFM1         ;INIT SIO WITH ID ADDRESS MARK AS SYNCH CHAR
         LD      E,IDFM2
         CALL    SIOINIT
         CALL    RXENABL
         LD      HL,DECTBL       ;SET H TO DECODE TABLE TOP ADDRESS
         LD      DE,USRMEM       ;PUT ID BYTES IN THIS TEMP USER MEMORY
         LD      BC,4            ;NUMBER BYTES TO READ (TRK, 00, SEC, 00)
         PUSH    IX              ;PRESERVE IX FOR DATA CRC BYTES
         LD      IX,IDCRCS       ;PUT ID CRC BYTES HERE
         CALL    RCEIV           ;BRING THE BYTES INTO MEMORY
         CALL    STOP            ;DISABLE WAIT STATE GENERATOR
         POP     IX              ;RESTORE DATA CRC BYTE POINTER

        ;SEE IF ID IS CORRECT
FMIDCK:  LD      HL,USRMEM       ;POINT TO WHAT SHOULD BE ID ADDRESS MARK (FE)
         LD      A,(TRACK)
         CP      (HL)            ;SEE IF TRACK MATCHES
         JR      NZ,FNDFSC2      ;IT DOESN'T MATCH, START OVER
         INC     HL
         LD      A,(HL)          ;GET WHAT SHOULD BE THE SIDE FLAG
         CP      SIDEFLG         ;
         JR      NZ,FNDFSC2
         INC     HL
         LD      A,(PHYSSEC)     ;GET PHYSICAL SECTOR NUMBER
         CP      (HL)            ;SEE IF MATCHES WHAT'S ON DISK
         JR      NZ,FNDFSC2      ;DOESN'T MATCH
         RET                     ;DAA, DAA, FOUND IT!
```

```
;*** COMPUTE SKEW ***
;(SKWTBL) MUST HAVE ADDRESS OF SKEW CONVERSION TABLE
;E IS THE LOGICAL SECTOR NUMBER
;PHYSICAL SECTOR NUMBER WILL RETURN IN E

COMPSKW:  LD      D,0
          DEC     E               ;SO SECTOR 1 WILL HAVE FIRST TABLE ENTRY
          LD      HL,(SKWTBL)
          ADD     HL,DE           ;COMPUTE TABLE ADDRESS
          LD      E,(HL)          ;GET SKEWED SECTOR NUMBER
          RET

;*** POST FILLER ***
;WAIT TILL AFTER FILL BYTES, THEN RETURN
PSTFILR:  LD      D,FILL1         ;INIT SIO WITH ENCODED FILL BYTE (4E)
          LD      E,FILL2
          CALL    SIOINIT         ;INIT SIO WITH ZERO AS ITS SYNCH CHAR
          LD      B,3             ;FIND THREE IN A ROW TO ENSURE THEY ARE THERE
          CALL    RXENABL
FILP1:    IN      A,(SIO_D)
          CP      FILL1
          JR      NZ,PSTFILR      ;NOT CONSEQUETIVE, START OVER
          IN      A,(SIO_D)
          CP      FILL2           ;NOT CONSEQUETIVE, START OVER
          JR      NZ,PSTFILR
          DJNZ    FILP1

          LD      B,16
FILP2:    IN      A,(SIO_D)       ;PASS UP 8 BYTES PRIOR TO SEARCHING FOR
          DJNZ    FILP2           ;A BYTE THAT IS NOT A FILL BYTE
FILP3:    IN      A,(SIO_D)       ;WAIT TILL FIRST NON FILL BYTE PRIOR TO
          CP      FILL1           ;RETURNING, PROBABLE A ZERO (DOESN'T MATTER)
          JR      NZ,FILR4
          IN      A,(SIO_D)
          CP      FILL2
          JR      Z,FILP3
FILR4:    CALL    STOP            ;STOP THE WAIT STATE GENERATOR
          RET
```

```
;*************************** INDX PULSE WAIT **********************************
;NOTE: THIS ROUTINE IS NOT NEEDED BY THE LIBRARY SOFT SECTOR ROUTINES
;SECTORS CAN BE FOUND QUICKER WITHOUT WAITING FOR THE IX PULSE
;THIS IS REQUIRED BY HARD SECTOR FORMATS
;A RETURN FROM THIS ROUTINE WILL BE EXECUTED AT THE RISING EDGE...
;OF THE INDEX PULSE

IXPLS::  CALL    IXLOW           ;WAIT TILL IX PULSE IS INACTIVE, SO CAN...
         LD      BC,50           ;DELAY TO LET PULSE FULLY TRANSITION
         CALL    DELAY
         CALL    IXHIGH          ;...BE SURE FIND LEADING EDGE, THEN WAIT...
         RET                     ;...FOR IT TO GO ACIVE

IXLOW:   IN      A,(PIOB_D)      ;GET DATA FROM PIO
         BIT     0,A             ;IF BIT 0 SET THEN HAVE IX PULSE
         JR      NZ,IXLOW        ;WAIT TILL BIT GOES LOW (INACTIVE) SO CAN
         RET                     ;ENSURE WE HAVE THE LEADING EDGE

IXHIGH:  IN      A,(PIOB_D)      ;GET DATA FROM PIO
         BIT     0,A             ;IF BIT 0 SET THEN HAVE IX PULS
         JR      Z,IXHIGH        ;WAIT FOR IX PULSE TO GO ACTIVE
         RET                     ;RETURN ON LEADING EDGE OF IX PULSE

;************************** SECTOR PULSE WAIT *********************************
;A RETURN FROM THIS ROUTINE WILL BE EXECUTED JUST AFTER THE PULSE PRIOR...
;TO THE CORRECT SECTOR PULSE. IF (SECTOR) IS ONE, A RET WILL OCCUR AT...
;THE LEADING EDGE OF IX PULSE.
;TO FIND THE EXACT POSITION OF THE DESIRED SECTOR, CALL HIGHP

FNDHSEC: CALL    IXPLS
         LD      A,(SECTOR)      ;GET DESIRED SECTOR #
         LD      D,A             ;USE D AS A SECTOR COUNTER
```

```
SECLP:   DEC     D               ;SEE IF THIS IS THE CORRECT SECTOR
         RET     Z               ;NEXT PULSE IS THE ONE LOOKING FOR
         LD      BC,50           ;DELAY TO LET PULSE FULLY TRANSITION
         CALL    DELAY
         CALL    HIGHP           ;WAIT FOR LEADING EDGE OF SECTOR PULSE
         LD      BC,50           ;DELAY TO LET PULSE FULLY TRANSITION
         CALL    DELAY
         CALL    LOWP            ;WAIT FOR TRAILING EDGE OF SECTOR PULSE
         JR      SECLP

HIGHP::  IN      A,(PIOB_D)      ;WAIT FOR SECTOR PULSE TO GO HIGH
         BIT     1,A             ;IF SET, THEN SECTOR PULSE IS ACTIVE
         JR      Z,HIGHP         ;WAIT FOR PULSE TO GO HIGH
         RET
LOWP:    IN      A,(PIOB_D)      ;WAIT FOR SECTOR PULSE TO GO HIGH
         BIT     1,A             ;IF BIT 1 SET, THEN SECTOR PULSE IS ACTIVE
         JR      NZ,LOWP         ;WAITING FOR LOW, SO CAN FIND LEADING EDGE
         RET

;************************ SIO ROUTINES ****************************************
RXENABL:: LD     A,3             ;POINT TO WRITE REGISTER 3
         OUT     (SIO_C),A
         LD      A,RX            ;RX ENABLE CODE
         OUT     (SIO_C),A
         LD      A,1             ;POINT TO WRITE REGISTER 1
         OUT     (SIO_C),A
         LD      A,WAITRX        ;ENABLE WAIT STATE GENERATOR FOR RX
         OUT     (SIO_C),A
         RET

TXENABL:: LD     A,5             ;POINT TO WRITE REGISTER 5
         OUT     (SIO_C),A
         LD      A,TX            ;TXMIT ENABLE CODE
         OUT     (SIO_C),A
         LD      A,1             ;POINT TO WRITE REGISTER 1
         OUT     (SIO_C),A
         LD      A,WAITTX        ;WORD TO ENABLE WAIT STATE GENERATOR
         OUT     (SIO_C),A
         RET
```

```
SIOINIT:: LD      A,RESET         ;RESET THE SIO
          OUT     (SIO_C),A
          LD      A,4             ;POINT TO WRITE REGISTER 4
          OUT     (SIO_C),A
          LD      A,SYNMODE       ;ENABLE THE SYNCH MODE IN THE SIO
          OUT     (SIO_C),A
          LD      A,6             ;POINT TO WRITE REGISTER 6
          OUT     (SIO_C),A
          LD      A,D             ;SET THE HIGH BYTE OF SYNCH CHARACTER
          OUT     (SIO_C),A
          LD      A,7             ;POINT TO WRITE REGISTER 7
          OUT     (SIO_C),A
          LD      A,E             ;SET THE LOW BYTE OF SYNCH CHARACTER
          OUT     (SIO_C),A
          RET

;TURN OFF THE WAIT STATE GENERATOR
STOP::    LD      A,1             ;POINT TO SIO WRITE REG 1
          OUT     (SIO_C),A
          LD      A,WAITOFF       ;WORD TO DISABLE WAITSTATE GENERATOR
          OUT     (SIO_C),A
          RET

;**************************** ERROR ROUTINES ********************************
RESEEK::  CALL    HOME            ;PUT HEAD OVER TRK 0, ZERO OUT TRK REGISTER
          CALL    SEEK            ;PUT HEAD OVER (TRACK), UPDATE TRK REGISTER
          RET

SEEKERR:: LD      A,0FEH          ;SEEK ERROR FLAG
          LD      (ERROR),A
          JP      ABORT

HANGUP::  LD      HL,REHMCNT      ;REHOME COUNTER
          DEC     (HL)
          JP      Z,SEEKERR       ;8 REHOMES WON'T FIX IT, ABORT
          CALL    RESEEK          ;REHOME AND SEEK (TRACK) AGAIN
          LD      A,52            ;REINIT HANGUP COUNTER
          LD      (HANGCNT),A
          RET                     ;GO BACK AND LOOK FOR CORRECT SECTOR AGAIN
```

```
;************************** CRC ROUTINES **************************************
;SUBROUTINE TO COMPUTE CRCS FOR UP TO 128 BLOCKS OF DATA
;GENENERATES A 16 BIT CRC VALUE USING THE POLYNOMIAL: X^16 + X^12 + X^5 + 1
;DATA BLOCKS CAN BE ANY SIZE UP TO 64K BYTES
;ENTER WITH REGISTERS SET ACCORDINGLY:
;(DTA) = ADDRESS OF FIRST BYTE OF FIRST BLOCK
;(NSECS) = NUMBER OF BLOCKS
;(BYTSEC) = NUMBER OF BYTES PER BLOCK
;IX = INITIAL CRC VALUE (SHOULD BE 0E295H FOR A DATA BLOCK HAVING...
;.....ADDRESS MARKS 'A1 A1 A1 FB' PRIOR TO DISK SECTOR DATA
;CRC VALUES ARE PLACED IN MEMORY STARTING AT CRCPTR

;********** COMPUTE CRC ***********

CMPCRCS:: CALL    CRCINIT         ;FILL IN CRC TABLE IF IT NEEDS IT
          LD      HL,(DTA)        ;BYTE POINTER
          LD      IY,CRCPTR       ;(IY) WILL RX CRCS OF EACH SECTOR
          LD      DE,(BYTSEC)     ;BYTE COUNTER
          LD      A,(NSECS)
          LD      B,A             ;SECTOR COUNTER

CRCBLKLP: PUSH    DE              ;SAVE BYTES PER BLOCK
          CALL    BLKCOMP         ;COMPUTE CRC VALUE FOR ONE BLOCK (SECTOR)
          LD      (IY+0),D        ;DE IS THE CRC VALUE
          INC     IY
          LD      (IY+0),E        ;STORE IT IN MEMORY FOR LATER USE BY CALLER
          INC     IY
          POP     DE              ;RESTORE BYTES PER BLOCK
          DJNZ    CRCBLKLP        ;LOOP TILL ALL BLOCKS HAVE A CRC
          RET                     ;DONE

;*** COMPUTE CRC VALUE FOR ONE BLOCK OF DATA
BLKCOMP:  EXX                     ;SET UP ALT REGS FOR CRC COMPUTATION
          PUSH    IX              ;PUT INITIAL CRC VAL IN DE VIA STACK
          POP     DE              ;DE' HOLDS CURRENT CRC VALUE
          LD      B,0
          EXX
```

```
DATALP:  LD      A,(HL)

;***********
UPDCRC:  EXX                     ;ALT REGS ARE SET UP FOR CRC COMPUTATION
         LD      HL,CRCTBL
         XOR     D
         LD      C,A
         ADD     HL,BC
         LD      A,(HL)
         XOR     E
         LD      D,A
         INC     H
         LD      E,(HL)
         EXX
;***********

         INC     HL              ;GET READY FOR NEXT DATA BYTE
         DEC     DE              ;SEE IF MORE DATA TO SEND TO UPDCRC
         LD      A,D
         OR      E
         JR      NZ,DATALP

         ;RETURN WITH CRC VALUE IN DE
         EXX                     ;GET CRC VALUE FROM DE'
         PUSH    DE
         EXX
         POP     DE
         RET
```

```
;*** CHECK CRC ***
;
CRCCHK::  CALL    CRCINIT         ;INITIALIZE FOR CRC COMPUTATIONS
          LD      HL,(DTA)        ;BYTE POINTER
          LD      IY,CRCPTR       ;IY IS PTR TO CRCS OF EACH SECTOR
          LD      DE,(BYTSEC)     ;BYTE COUNTER
          LD      A,(NSECS)
          LD      B,A             ;SECTOR COUNTER

CHKLOOP:  PUSH    DE              ;SAVE BYTES PER BLOCK
          CALL    BLKCOMP         ;COMP CRC FOR BYTES IN MEMORY
          LD      A,(IY+0)        ;GET FIRST CRC BYTE FROM DISK
          INC     IY              ;POINT TO FOLLOWING BYTE FROM DISK
          CP      D               ;SEE IF DISK VALUE = COMPUTED VALUE
          JR      Z,CHK2          ;OK, CONTINUE
          CALL    CRCER           ;HAVE A CRC ERROR, INC ERROR COUNTER
          JR      SKIP2


CHK2:     LD      A,(IY+0)        ;GET SECOND BYTE OF CRC FROM DISK
          CP      E
          CALL    NZ,CRCER        ;INCREMENT CRC ERROR COUNTER

SKIP2:    INC     IY              ;POINT TO NEXT CRC BYTE
          POP     DE              ;RESTORE BYTES PER BLOCK
          DJNZ    CHKLOOP         ;CONTINUE TILL ALL BLOCKS/SECTORS ARE CHKED
          LD      A,(CRCERRS)     ;RET WITH NUMBER OF CRC ERRORS
          RET

CRCER:    LD      A,(CRCERRS)     ;INCREMENT NUMBER OF CRC ERRORS
          INC     A
          LD      (CRCERRS),A
          RET
```

```
;********** INIT FOR CRC COMPUTATION *********
;
CRCINIT: XOR     A               ;INIT # OF CRC ERRORS TO ZERO
         LD      (CRCERRS),A
         LD      A,(TBLFLAG)     ;IF FF THEN CRC TABLE HAS NOT BEEN FILLED IN
         CP      0
         RET     Z               ;TABLE ALREADY FILLED IN, SKIP THE CRCINIT
         XOR     A               ;CLEAR TABLE FLAG TO SHOW ITS NOW FILLED IN...
         LD      (TBLFLAG),A     ;THEN FILL IN THE TABLE:

         LD      HL,CRCTBL
         LD      C,0
GLOOP:   EX      DE,HL
         LD      HL,0
         LD      A,C
         PUSH    BC
         LD      B,8
         XOR     H
         LD      H,A
LLOOP:   ADD     HL,HL
         JR      NC,LSKIP
         LD      A,10H
         XOR     H
         LD      H,A
         LD      A,21H
         XOR     L
         LD      L,A
LSKIP:   DJNZ    LLOOP
         POP     BC
         EX      DE,HL
         LD      (HL),D
         INC     H
         LD      (HL),E
         DEC     H
         INC     HL
         INC     C
         JR      NZ,GLOOP
         RET
```

```
;************************** CRC MFM ENCODE ********************************
;ENCODES CRC BYTES FOR ONE SECTOR, SAVES IN (IY), DOESN'T CHANGE IY
;BYTES TO ENCODE ARE POINTED TO BY IX  /  FIRST BYTE OF SECTOR IS IN (DTA)
;NUMBER OF BYTES PER SECTOR ARE IN (BYTSEC)
;IX IS INCREMEMTED BY 2 WHICH IS NEEDED BY LIBRARY ROUTINES
CRCENCOD: LD      IY,ENCCRC       ;PUT ENCODED CRC BYTES HERE
          LD      BC,(BYTSEC)     ;BYTES PER SECTOR
          LD      HL,(DTA)
          ADD     HL,BC           ;POINT TO LAST BYTE OF SECTOR
          DEC     HL
          LD      A,(HL)
          RRA                     ;GET LAST BIT OF LAST BYTE OF SECTOR AND...
          PUSH    AF              ;SAVE IT
          LD      DE,DDENTBL      ;DOUBLE DENSITY ENCODING TABLE
          LD      B,2             ;NUMBER OF CRC BYTES
          LD      C,00011111B     ;MASK BYTE
CRCEN2:   POP     AF              ;GET LAST DATA BIT OF LAST BYTE
          LD      A,(IX+0)
          RRA
          PUSH    AF              ;SAVE FOR NEXT LOOP
          RRA                     ;COMPUTE TABLE ADDRESS
          RRA
          RRA
          AND     C
          LD      E,A
          LD      A,(DE)          ;GET ENCODED HIGH ORDER NIBBLE
          LD      (IY+0),A        ;SAVE IT
          INC     IY
          LD      A,(IX+0)        ;GET SAME BYTE AGAIN
          INC     IX              ;POINT TO FOLLOWING BYTE
          AND     C               ;COMPUTE TABLE ADDRESS FOR LOW NIBBLE
          LD      E,A
          LD      A,(DE)          ;GET ENCODED LOW NIBBLE
          LD      (IY+0),A
          INC     IY
          DJNZ    CRCEN2
          POP     AF              ;CLEAN STACK
          LD      IY,ENCCRC       ;PTR TO FIRST ENCODED CRC BYTE
          RET
```

```
;***************************** ABORT **************************************
;

ABORT::  CALL    DSELECT         ;DESELECT ALL DRIVES
         LD      SP,(STKPTR)     ;RESET THE ORIGINAL STACK POINTER
         LD      A,(ERROR)
         RET                     ;RETURN TO MAIN ROUTINE 'C'

         .DEPHASE
         END
```

```
;*****************************************************
;*                                                   *
;*      DATE:  28 OCT 83                             *
;*      FILE NAME: IBM.MAC                           *
;*      PURPOSE: READ AND WRITE AN UNLIMITED NUMBER OF  *
;*               CONTIGUOUS SECTORS TO AN IBM 8" SINGLE *
;*               DENSITY FORMATED DISK.              *
;*****************************************************


INCLUDE LIBADDR.THF                 ;ADDRESSES FOR UDCLIB ROUTINES

SECPTRK   EQU     26                ;SECTORS PER TRACK
BYTPSEC   EQU     128               ;BYTES PER SECTOR
SIZ_DEN   EQU     00001000B         ;RESET BIT 4 FOR SINGLE DENSITY, SET BIT...
                                    ;3 FOR 8"
TEXTURE   EQU     0                 ;SOFT SECTORED
SKEW      EQU     1                 ;USE IBM STANDARD 8" SKEW
FSECNUM   EQU     1                 ;FIRST SECTOR NUMBER IN ID FIELD

          ;MISELLANEOUS STORAGE (57E0-57FF IS RESERVED FOR THIS PURPOSE)
CRCADDR   EQU     57E0H             ;HOLD ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
NCRCBLKS  EQU     57E2H             ;NUM CRC BLOCKS (SAVE AS SECTOR)


          .Z80

;FOR BOTH A READ AND A WRITE, THE REGISTERS MUST BE AS SHOWN BELOW

;"TRANSIT" SETS THE FOLLOWING PARAMETERS
;HL - DTA
;A  - 1 for a write, else read
;B  - track
;C  - start sector
;D  - drive
;E  - number of byte sectors to txfer

;************************ CALL RELATIVE MACRO ******************************

CALREL    MACRO   ADDRESS
          CALL    PSHRET            ;PUT THE RETURN ADDRESS IN STACK
          JR      ADDRESS           ;JUMP RELATIVE TO ROUTINE
          ENDM
;**************************************************************************
```

```
        ASEG
        ORG     100H

IBM:    JR      START

        ;PARAMETERS NEEDED BY THIEF.C
        DB      1               ;RECORDS PER SECTOR
        DB      26              ;RECORDS PER TRACK
        DB      52              ;OFFSET (#RECORDS PRIOR TO DIRECTORY)
        DB      16              ;# OF RECORDS IN DIRECTORY
        DW      241             ;# OF K AVAILABLE FOR FILE STORAGE ON DISK
                                ;***** MUST USE TWO BYTES ******


START:  LD      (STKPTR),SP     ;SAVE FOR NESTED ABORTS IN UDCLIB
        PUSH    AF              ;SAVE RD_WRT COMMAND
        CALREL  INIT            ;INITIALIZE, SELECT DRIVE ETC.
        POP     AF              ;RESTORE RD_WRT COMMAND
        DEC     A               ;SEE IF READ OR WRITE IS WANTED
        JR      Z,WRITE         ;WRITE TO DISK
        JR      READ

;INITIALIZE PARAMETERS NEEDED BY UDC LIBRARY ROUTINES
INIT:   LD      (DTA),HL        ;DISK TRANSFER ADDRESS
        LD      (CRCADDR),HL    ;ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
        LD      A,D             ;DRIVE #
        LD      (DRIVE),A       ;
        LD      A,B             ;START TRACK #
        LD      (TRACK),A
        LD      A,C             ;SECTOR #
        LD      (SECTOR),A      ;SAVE SECTOR #
        LD      A,E             ;# SECTORS
        LD      (NSECS),A       ;PHYSICAL SECTOR SIZE
        LD      (NCRCBLKS),A    ;ONE CRC COMPUTATION FOR EACH SECTOR
        LD      A,SKEW          ;0 IF NO SKEW, 1 IF SKEW NEEDED / IF SKEW...
        LD      (SKWFLG),A      ;...IS NEEDED, THEN NEED TO SET SKEW TABLE...
        LD      A,FSECNUM       ;FIRST SECTOR NUMBER
        LD      (FRSTSEC),A
        LD      (SKWTBL),HL     ;IBM 3740 FM STANDARD SKEWING
        LD      A,SECPTRK       ;SECTORS PER TRACK
        LD      (SEC_TRK),A     ;
        LD      HL,BYTPSEC      ;BYTES PER SECTOR
        LD      (BYTSEC),HL
        LD      A,TEXTURE       ;IBM IS SOFT SECTORED DISK
        LD      (HD_SOFT),A
        XOR     A               ;INIT ERROR FLAG
        LD      (ERROR),A
        LD      A,SIZ_DEN       ;SIZE & DENSITY PARAMETER NEEDED BY DRVSET
        CALL    DRVSET          ;SELECT DRIVE INDICATED BY (DRIVE)
        RET
```

```
;****************************** READ ****************************************
READ:   LD      HL,RDSFM        ;ADDRESS OF LIBRARY ROUTINE TO READ A...
        LD      (ROUTADR),HL    ;SECTOR USING THE IBM FM STANDARD
        CALL    TXFER           ;GO DO IT TO IT
        LD      A,(ERROR)
        CP      0
        RET     NZ              ;RETURN WITH ACC = ERROR FLAG...
                                ;FF = BAD FORMAT, FE = SEEK ERROR
        LD      HL,(CRCADDR)    ;SET UP FOR CRC CHECK
        LD      (DTA),HL
        LD      A,(NCRCBLKS)
        LD      (NSECS),A
        LD      IX,0BF84H       ;INITIAL CRC VALUE (STANDARD FOR FM)
        CALL    CRCCHK          ;RET WITH NUM CRC ERRORS IN ACC
        RET

;****************************** WRITE ***************************************
WRITE:  LD      IX,0BF84H       ;INITIAL CRC VALUE (STANDARD FOR FM)
        CALL    CMPCRCS         ;COMPUTE CRCS, PLACE VALUES IN (CRCPTR)
        LD      HL,WRTSFM       ;ADDRESS OF LIB ROUTINE TO WRITE AN FM...
        LD      (ROUTADR),HL    ;...SECTOR USING IBM FM STANDARD
        CALL    TXFER
        LD      A,(ERROR)       ;RET WITH ANY POSSIBLE ERROR FLAGS IN ACC
        RET

        END
```

```
;*****************************************************
;*                                                   *
;*     DATE:  28 OCT 83                              *
;*     FILE NAME: NS.MAC                             *
;*     PURPOSE: READ AND WRITE AN UNLIMITED NUMBER OF  *
;*              CONTIGUOUS SECTORS TO A NORTHSTAR    *
;*              HORIZON SINGLE DENSITY DISK          *
;*****************************************************


INCLUDE LIBADDR.THF               ;ADDRESSES OF UDC LIBRARY ROUTINES

SECPTRK   EQU     10              ;SECTORS PER TRACK
BYTPSEC   EQU     256             ;BYTES PER SECTOR
SIZ_DEN   EQU     00000000B       ;RESET BIT 4 FOR SINGLE DENSITY, RESET BIT...
                                  ;3 FOR 5 "
TEXTURE   EQU     1               ;HARD SECTORED
SKEW      EQU     0               ;NO SKEW
FSECNUM   EQU     1               ;FIRST SECTOR NUMBER IN ID FIELD (0 OR 1)

          ;MISELLANEOUS STORAGE (57E0 - 57FF IS RESERVED FOR THIS PURPOSE)
CRCADDR   EQU     57E0H           ;HOLD ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
NCRCBLKS  EQU     57E2H           ;NUM CRC BLOCKS (SAME AS NUM SECTORS)
CRCERS    EQU     57E3H           ;NUMBER OF CRC ERRORS


;FOR BOTH A READ AND A WRITE, THE REGISTERS MUST BE AS SHOWN BELOW

;"C" PROGRAM SETS THE FOLLOWING PARAMETERS
;HL - DTA
;A  - 1 for a write, else read
;B  - track
;C  - start sector
;D  - drive
;E  - number of *128 byte sectors to tx

;*********************** CALL RELATIVE MACRO ********************************
CALREL    MACRO   ADDRESS
          CALL    PSHRET          ;PUT CORRECT RET ADDRESS IN STACK
          JR      ADDRESS         ;JUMP RELATIVE TO ROUTINE
          ENDM
;****************************************************************************
```

```
        .Z80
        ASEG
        ORG     100H

NS:     JR      START

        ;PARAMETERS NEEDED BY TRANSIT.C
        DB      2                  ;RECORDS PER SECTOR
        DB      20                 ;RECORDS PER TRACK
        DB      60                 ;OFFSET (# RECORDS PRIOR TO DIRECTORY)
        DB      16                 ;# RECORDS IN DIRECTORY
        DW      78                 ;# OF K AVAILABLE FOR FILE STORAGE ON DISK
                                   ;***** MUST USE TWO BYTES ******

START:  LD      (STKPTR),SP        ;SAVE FOR NESTED ABORTS IN UDCLIB
        PUSH    AF                 ;SAVE RD_WRT COMMAND
        CALREL  INIT               ;INITIALIZE, SELECT DRIVE ETC.
        POP     AF                 ;RESTORE RD_WRT COMMAND
        DEC     A                  ;SEE IF READ OR WRITE IS WANTED
        JR      Z,WRITE            ;WRITE TO DISK
        JR      READ

;INITIALIZE PARAMETERS NEEDED BY CONTROLLER ROUTINES
INIT:   LD      (DTA),HL           ;DISK TRANSFER ADDRESS
        LD      (CRCADDR),HL       ;ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
        LD      A,D                ;DRIVE #
        LD      (DRIVE),A          ;
        LD      A,B                ;START TRACK #
        LD      (TRACK),A
        LD      A,C                ;SECTOR #
        LD      (SECTOR),A
        LD      A,E                ;NUMBER SECTORS
        LD      (NSECS),A          ;
        LD      (NCRCBLKS),A       ;ONE CRC COMPUTATION FOR EACH SECTOR
        LD      A,SKEW             ;0 IF NO SKEW, 1 IF SKEW NEEDED / IF SKEW...
        LD      (SKWFLG),A         ;...IS NEEDED, THEN NEED TO SET SKEW TABLE...
                                   ;...ADDRESS
        LD      A,SECPTRK          ;SECTORS PER TRACK
        LD      (SEC_TRK),A        ;
        LD      HL,BYTPSEC         ;BYTES PER SECTOR
        LD      (BYTSEC),HL
        LD      A,TEXTURE          ;NORTHSTAR USES HARD SECTORED DISK
        LD      (HD_SOFT),A
        XOR     A                  ;INITIALIZE ERROR FLAG
        LD      (ERROR),A
        LD      A,FSECNUM          ;FIRST SECTOR NUMBER
        LD      (FRSTSEC),A
        LD      A,SIZ_DEN          ;PARAMETER NEEDED BY DRVSET
        CALL    DRVSET             ;SELECT DRIVE INDICATED BY (DRIVE)
        RET
```

```
READ:    LD      HL,RDHDFM       ;ADDRESS OF LIBRARY ROUTINE TO READ A...
         LD      (ROUTADR),HL    ;SECTOR USING THE IBM MFM STANDARD
         CALL    TXFER           ;GO DO IT TO IT
         CALREL  NSCRCHK         ;NON STANDARD CRC CHECK FOR NS / RET WITH...
         RET                     ;NUM CRC ERROR IN A

WRITE:   CALREL  NSCRCMP         ;COMPUTE CRCS, PLACE VALUES IN (CRCPTR)
         LD      HL,WRHDFM       ;ADDRESS OF LIB ROUTINE TO WRITE HARD FM...
         LD      (ROUTADR),HL    ;...SECTOR
         CALL    TXFER
         RET

;************************* NS CRC COMPUTATIONS *****************************
NSCRCMP: LD      DE,CRCPTR       ;GLOBAL POINTER TO CRC BYTES
         LD      HL,(CRCADDR)    ;SAME AS DISK TRANSFER ADDRESS
         LD      A,(NCRCBLKS)    ;SAME AS NSECS
         LD      C,A
CRCLP:   XOR     A               ;INITIAL CRC VALUE IS ZERO
         LD      B,A             ;BYTE COUNTER (256 BYTES PER BLOCK/SECTOR)
         CALREL  BLKCOMP
         LD      (DE),A          ;STOR FIRST CRC BYTE
         INC     DE              ;NS USES ONLY ONE CRC BYTE/LIBRARY ROUTINES...
         INC     DE              ;...ASSUME TWO SO PUT IN MEM EVERY OTHER BYTE
         DEC     C               ;BLOCK COUNTER
         JR      NZ,CRCLP
         RET
```

```
NSCRCHK:  LD      DE,CRCPTR       ;POINTER TO FIRST CRC BYTE
          LD      HL,(CRCADDR)
          LD      A,(NCRCBLKS)    ;NUMBER CRC BLOCKS/SECTORS
          LD      C,A
          XOR     A               ;INITIALIZE NUMBER CRC ERRORS TO ZERO
          LD      (CRCERS),A

CHKLOOP:  XOR     A
          LD      B,A
          CALREL  BLKCOMP         ;COMPUTE CRC FOR WHAT WAS READ
          EX      DE,HL
          CP      (HL)            ;SEE IF MATCHES CRC BYTE ON DISK
          EX      DE,HL
          JR      Z,SKIPERR       ;NO ERROR IF MATCH

CRCERR:   LD      A,(CRCERS)      ;INCREMENT NUMBER OF CRC ERRORS
          INC     A
          LD      (CRCERS),A

SKIPERR:  INC     DE              ;POINT TO NEXT CRC BYTE FROM DISK
          INC     DE
          DEC     C               ;BLOCK/SECTOR COUNTER
          JR      NZ,CHKLOOP
          LD      A,(CRCERS)      ;RETURN WITH NUMBER CRC ERRORS IN ACC
          RET

BLKCOMP:  XOR     (HL)            ;NON STANDARD CRC COMPUTATION FOR NORTH STAR
          RLCA
          INC     HL              ;BYTE POINTER
          DJNZ    BLKCOMP         ;256 BYTES PER CRC BLOCK
          RET

          END
```

```
;*************************************************************
;*                                                           *
;*      DATE:  28 OCT 83                                     *
;*      FILE NAME: NEC.MAC                                   *
;*      PURPOSE: READ AND WRITE AN UNLIMITED NUMBER OF       *
;*               CONTIGUOUS SECTORS TO A NEC 8000 DISK       *
;*************************************************************

INCLUDE LIBADDR.THF                     ;ADDRESSES OF LIBRARY ROUTINES

SECPTRK   EQU     16                    ;SECTORS PER TRACK
BYTPSEC   EQU     256                   ;BYTES PER SECTOR
SIZ_DEN   EQU     00010000B             ;SET BIT 4 FOR DOUBLE DENSITY, RESET BIT...
                                        ;3 FOR 5 "
TEXTURE   EQU     0                     ;SOFT SECTORED
SKEW      EQU     0                     ;NO SKEW
FSECNUM   EQU     1                     ;FIRST SECTOR NUMBER IN ID FIELD (0 OR 1)

          ;MISELLANEOUS STORAGE (57E0 - 57FF IS RESERVED FOR THIS PURPOSE)
CRCADDR   EQU     57E0H                 ;HOLD ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
NCRCBLKS  EQU     57E2H                 ;NUM CRC BLOCKS (SAVE AS SECTOR)

         .Z80

;FOR BOTH A READ AND A WRITE, THE REGISTERS MUST BE AS SHOWN BELOW
;"C" PROGRAM SETS THE FOLLOWING PARAMETERS
;HL - DTA
;A  - 1 for a write, else read
;B  - track
;C  - start sector
;D  - drive
;E  - number of byte sectors to txfer

;*********************** CALL RELATIVE MACRO ******************************

CALREL    MACRO   ADDRESS
          CALL    PSHRET
          JR      ADDRESS
          ENDM
;**************************************************************************
```

```
        ASEG
        ORG     100H
;NEC::
SYSTEM:: JR     START

        ;PARAMETERS NEEDED BY TRANSIT.C
        DB      2               ;RECORDS PER SECTOR
        DB      32              ;RECORDS PER TRACK
        DB      64              ;OFFSET (DIRECTORY IS 65TH RECORD ON DISK)
        DB      16              ;# OF RECORDS IN DIRECTORY
        DW      150             ;# OF K AVAILABLE FOR FILE STORAGE ON DISK
                                ;***** MUST USE TWO BYTES ******


START:  LD      (STKPTR),SP     ;SAVE FOR NESTED ABORTS IN UDCLIB
        PUSH    AF              ;SAVE RD_WRT COMMAND
        CALREL  INIT            ;INITIALIZE, SELECT DRIVE ETC.
        POP     AF              ;RESTORE RD_WRT COMMAND
        DEC     A               ;SEE IF READ OR WRITE IS WANTED
        JR      Z,WRITE         ;WRITE TO DISK
        JR      READ

;INITIALIZE PARAMETERS NEEDED BY CONTROLLER ROUTINES
INIT:   LD      (DTA),HL        ;DISK TRANSFER ADDRESS
        LD      (CRCADDR),HL    ;ADDRESS OF 1ST BYTE TO COMPUTE CRC FOR
        LD      A,D             ;DRIVE #
        LD      (DRIVE),A       ;
        LD      A,B             ;START TRACK #
        LD      (TRACK),A
        LD      A,C             ;SECTOR #
        LD      (SECTOR),A
        LD      A,E             ;NUMBER CONTIGUOUS SECTORS TO TXFER
        LD      (NSECS),A       ;PHYSICAL SECTOR SIZE
        LD      (NCRCBLKS),A    ;ONE CRC COMPUTATION FOR EACH SECTOR

        LD      A,SKEW          ;0 IF NO SKEW, 1 IF SKEW NEEDED / IF SKEW...
        LD      (SKWFLG),A      ;...IS NEEDED, THEN NEED TO SET SKEW TABLE...
                                ;...ADDRESS
        LD      A,SECPTRK       ;SECTORS PER TRACK
        LD      (SEC_TRK),A     ;
        LD      HL,BYTPSEC      ;BYTES PER SECTOR
        LD      (BYTSEC),HL
        LD      A,TEXTURE       ;NEC IS SOFT SECTORED DISK
        LD      (HD_SOFT),A
        XOR     A               ;INITIALIZE ERROR CODE
        LD      (ERROR),A

        LD      A,FSECNUM       ;FIRST SECTOR NUMBER
        LD      (FRSTSEC),A
        LD      A,SIZ_DEN       ;PARAMETER NEEDED BY DRVSET
        CALL    DRVSET          ;SELECT DRIVE INDICATED BY (DRIVE)
        RET
```

```
;******************************* READ ****************************************
READ:   LD      HL,RDMFM        ;ADDRESS OF LIBRARY ROUTINE TO READ A...
        LD      (ROUTADR),HL    ;SECTOR USING THE IBM MFM STANDARD
        CALL    TXFER           ;GO DO IT TO IT
        LD      A,(ERROR)
        CP      0
        RET     NZ              ;RETURN WITH ACC = ERROR FLAG...
                                ;FF = BAD FORMAT, FE = SEEK ERROR
        LD      HL,(CRCADDR)    ;SET UP FOR CRC CHECK
        LD      (DTA),HL
        LD      A,(NCRCBLKS)
        LD      (NSECS),A
        LD      IX,0E295H       ;INITIAL CRC VALUE (STANDARD FOR MFM)
        CALL    CRCCHK          ;RET WITH NUM CRC ERRORS IN ACC
        RET

;******************************** WRITE *************************************
WRITE:  LD      IX,0E295H       ;INITIAL CRC VALUE (STANDARD FOR MFM)
        CALL    CMPCRCS         ;COMPUTE CRCS, PLACE VALUES IN (CRCPTR)
        LD      HL,WRTMFM       ;ADDRESS OF LIB ROUTINE TO WRITE AN MFM...
        LD      (ROUTADR),HL    ;...SECTOR USING IBM MFM STANDARD
        CALL    TXFER
        XOR     A               ;NO ERRORS FLAG TO TRANSIT.C
        RET

        END
```

VITA

Frank Nicholas Elam was born on 12 August 1952 in San Antonio, Texas. He graduted from high school in Charleston, South Carolina in 1970 and attended The Citadel from which he received the degree of Bachelor of Science in Electrical Engineering in May 1974. Upon graduation, he received a commission in the USAF through the ROTC program. He completed pilot training and received his wings in January 1976. He then served as an F-4 pilot, flight leader, and instructor pilot in the 614th Tactical Fighter Squadron, Torrejon, Spain and 68th Tactical Fighter Squadron, Moody AFB, Georgia. He entered the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 1407 Merrimac Street
Charleston, South Carolina 29406

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/CE/ENG/83D-20	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45385	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS.: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See box 19	

12. PERSONAL AUTHOR(S): Frank N. Elam, B.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
	FROM ____ TO ____	1983 November	172

16. SUPPLEMENTARY NOTATION

16 Feb 1984 Lyn Wolaver AFIT/NC

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: Universal Disk Controller for Microcomputers

Thesis Chairman: Charles W. Lillie, Major, USAF

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Charles W. Lillie, Major, USAF	22b. TELEPHONE NUMBER (Include Area Code): 513-255-5533; 22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Re[illegible] To [illegible] [illegible] Feb [illegible]

Attn (?): Capt. Frank N. E[illegible]
AFIT EN
WPAFB, OH 45433

Subject: Distribution Authorization

To: Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

I hereby authorize the Defense Technical Information Center to release my AFIT masters thesis entitled "Universal Disk Controller for Microcomputers" without any restriction.

Frank N Elam

Frank N. Elam, Capt USAF

FILM

4-8